1. Wysiwyg0100 Getting Started
2. Hosting PDF on OpenStax
3. Step-by-Step Guide to OpenStax Publishing
4. Authoring OpenStax Documents in Apache OpenOffice Writer

Wysiwyg0100 Getting Started
If you can create a valid XHTML file using a WYSIWYG editor such as the free version of Microsoft Expression Web 4, you can easily create and publish content on OpenStax using Dick Baldwin's free XHTML-to-CNXML translator program. This page explains how to use Baldwin's translator program along with a WYSIWYG XHTML editor to create and publish CNXML content on OpenStax.

Revised: Thu Mar 24 14:45:48 CDT 2016

Note:
This page is part of a Book titled OpenStax Publishing with a WYSIWYG Editor.

Table of contents

Preface

This page is part of the took titled OpenStax Publishing with a WYSIWYG Editor .

The elusive WYSIWYG CNXML editor

Have you ever wished that you could use a WYSIWYG editor to create and publish CNXML content on OpenStax. Well, that is what I was wishing for in 2009. Rather than just wish, however, I did something about it. I wrote an XHTML-to-CNXML translator program for translating XHTML files into CNXML files suitable for uploading and publishing on OpenStax. *(OpenStax was called Connections at that point in time.)*

If you go to the Advanced Search feature at OpenStax CNX and search for an author named Richard Baldwin, you will learn that I have used my translator program along with Microsoft Expression Web 4 to create and publish more than 770 Books and Pages on OpenStax since 2009. My approach really does work and it is easy to use.

How I create and publish CNXML content on OpenStax

These are the four steps that I use to create and publish CNXML content on OpenStax:

1. For each new Page, I use the free version of Microsoft Expression Web 4 to create a valid XHTML file describing my content. This mostly involves the use of the built-in WYSIWYG editor in Expression Web 4.
2. Then I process the XHTML file through my XHTML-to-CNXML translator program named **CNXMLprep12** to produce a CNXML file suitable for uploading to OpenStax.
3. Then I upload the CNXML file to OpenStax.
4. Last but not least, I publish the CNXML file on OpenStax.

It couldn't be simpler. I do everything at the XHTML level, mostly using the WYSIWYG editor. I rarely touch the raw XHTML code and I never touch the raw CNXML code.

I maintain all of my archives in XHTML format. When I need to update a page, *(as I am doing right now)* , I update the corresponding XHTML file, run it through my translator, upload the modified CNXML file to OpenStax,

and publish it. Once again, I never touch the raw CNXML code either before or after uploading it to OpenStax.

You too can publish CNXML content on OpenStax

In support of my strong interest in OER and free textbooks, I have decided to make my translator program available for use by the general public.

If you don't know anything about CNXML but you already know how to create a valid XHTML file using a WYSIWYG editor such as the free version of Microsoft Expression Web 4 , you can easily create and publish your CNXML content on OpenStax using my free XHTML-to-CNXML translator program.

If you don't know about XHTML but you do already know how to use Microsoft Word, you should have no difficulty learning how to use Microsoft Expression Web 4. That is because the user interface of Microsoft Expression Web 4 is very similar to the user interface of earlier generations of Microsoft Word.

By now you might be asking, *"Why not just create and upload Microsoft Word documents to OpenStax?"* Click here to learn my reasons for not doing that.

This page explains how to use my translator program to create and publish your content. This page also provides a link to a downloadable zip file that contains everything you need to get started *(including executable Java program files and an XHTML template file)* . The one thing that the zip file doesn't include is a WYSIWYG XHTML editor. However, as of March 2016, you can download the free version of Microsoft Expression Web 4 .

While you should be able to use any WYSIWYG XHTML editor, I recommend that you use Microsoft Expression Web 4 if possible because it is free, because it has a built-in validator, and because it is relatively easy to use.

By the way, this page was produced using the free version of Microsoft Expression Web 4 in conjunction with my translator program.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the Figures and Listings while you are reading about them.

Figures

Listings

Discussion

My translator program is named **CNXMLprep12** . As you can see from the number at the end of the name, it has gone through quite a few improvements and iterations since 2009.

How it works

This section contains a brief description of how the program works. If you don't care how it works, feel free to skip to the next section . If you are not interested in any background information at this point and you just want to

create, upload and publish some content on OpenStax, skip ahead to the section titled Run the program .

The program consists of three major Java classes:

- CNXMLprep12
- CNXMLFirstPass12
- CNXMLSecondPass12

The first class in the list is the driver.

Code in an object of the class named **CNXMLFirstPass12** reads a valid XHTML file and performs several cleanup actions to make it easier to convert the XHTML file to CNXML format. It writes a temporary output file named **CNXMLtemp.html** in XHTML format in the current folder.

Following that, code in an object of the class named **CNXMLSecondPass12** reads the file named **CNXMLtemp.html** and translates it into CNXML code suitable for uploading to OpenStax. *(See the section named Run the Program for instructions on how to create and upload a CNXML file to OpenStax.)*

Instructions regarding your XHTML code

I assume that you will be creating content containing headers, ordinary paragraph text, preformatted text, ordered lists, unordered lists, blockquote, tables, images, and programming source code, along with other kinds of structures commonly found on web pages. This section contains guidelines on how best to create that content in your XHTML editor.

Before going further, I need to point out that there are many CNXML tags that cannot be created by this program. This is not a complete WYSIWYG CNXML editor. Rather, it is a program designed to support a reasonable subset of approximately 57 tags that are available in CNXML.

Preformatted text and program source code

If you include program source code *(such as Java or C++ source code)* in the XHTML document, it must be placed in a preformatted block. For example, the code shown in Listing 1 was placed in a preformatted block in a table. It was placed in the table to make it easy to provide a caption and an anchor.

Listing 1 . Preformatted text.

```
out.println("<html xmlns=\"http://www.w3.org"
+

"/1999/xhtml\">");

out.println("<head>");
out.println("<meta content=\"text/html; " +
"charset=utf-8\" http-equiv=\"Content-
Type\"/>");

out.println("<title>dummy title</title>");
out.println("</head>");
out.println("<body>");<out.println("<body>");
```

Insert a footer at the top. It moves to the bottom.

Table footers are optional. Although they are actually created at the top *(below the header)* , they move to the bottom when the table is displayed in a browser.

There must not be any formatting code, such as **bold text** inside a preformatted block.

All left angle brackets, right angle brackets, quotes, and ampersands in preformatted text must be converted to the following entities in the raw XHTML code before attempting the **translation** into CNXML.

Note:

```
< = &lt;
> = &gt;
" = &quot;
& = &amp;
```

As an alternative to placing preformatted text in a table, free-standing preformatted text such as the source code shown below is also allowed. Just be sure to deal with the angle brackets, quotes, and ampersands as described above if you create free-standing preformatted text.

```
import org.newdawn.slick.AppGameContainer;
import org.newdawn.slick.BasicGame;
import org.newdawn.slick.GameContainer;
import org.newdawn.slick.Graphics;
import org.newdawn.slick.SlickException;

public class Slick0130a extends BasicGame{

  //Instance variables for use in computing and
  // displaying total time since program start and
  // time for each frame.
  double totalTime = 0;
```

```
  int incrementalTime = 0;

  public Slick0130a(){
    //Call to superclass constructor is required.
    super("Slick0130a, Baldwin.");
  }//end constructor
  //-------------------------------------------
------//
```

Hyperlinks

The program translates hyperlinks for local anchors *(inside the document)* and for external web sites.

All hyperlink target names must be unique. You must also make certain that target names for local anchors don't include spaces or punctuation characters such as comma, colon, semicolon, dash, etc. Just use letters, numbers, and underscore characters *(Figure 01 for example)* for your local anchor target names.

Table considerations

XHTML tables and CNXML tables come in many different varieties and formats. This program supports a subset of those varieties and formats.

The table "width" attribute

The XHTML table **width** attribute is translated into a CNXML **pgwide** attribute. Here is what the OpenStax documentation has to say about the **pgwide** attribute:

> **pgwide** (optional): Determines the available width of the table.

> Possible values:
>
> - **0** - Maximum width of the table is the galley width (default).
> - **any positive integer** - The width of the table is the entire width of the page.

What this really seems to mean is that for a **width** value of 0, the table will only be wide enough to display its contents. For a **width** value of 1, the table will be the full width of the OpenStax page in the browser.

As of March 16, this width variation only works in the Legacy view. When the table is viewed in the newer OpenStax view, all tables seem to be full width regardless of the value of **pgwide** . I assume that this is an error that will be fixed sometime in the future.

In this program, the table **width** value must be either 0 or 1. Percent (%) values are not allowed, nor are any values other than 0 or 1.

The table "border" attribute

Tables in the XHTML file must have **border** attribute values of either 15 or 20. No other values are allowed in this version of the program.

A **border** value of 15 results in a CNXML **note** object being created as shown by the two note objects in the Miscellaneous section.. *(A CNXML **note** object displays as the full width of the browser window regardless of the value of the **width** attribute.)*

A **border** value of 20 results in a CNXML **table** object as shown in Figure 1 .

Structure of a simple table

This program requires that all **table** objects have the full structure shown in Figure 1 except that the footer section is optional and can be omitted.

Figure 1. Required table structure.

```
table border="15 or 20" width="0 or 1"
  thead
    tr
      th
        Caption text
      /th
    /tr
  /thead

  tfoot - optional
    tr
      td
        Footer text
      /td
    /tr
  /tfoot

  tbody
    tr
      td
        Table content
      /td
    /tr
  /tbody
/table
```

The easiest way to create such a table is probably to copy a table from the template that I will provide, paste it into your WYSIWYG editor, and then edit as needed. However, you should be able to create an acceptable table using only the WYSIWYG features of your XHTML editor if you prefer not to use a template.

Although footers are created at the top, they are displayed at the bottom when the page is displayed in a browser.

Structure of multi-column, multi-row tables

I normally place my Figures and Listings in tables having a header, one column, one row, and optionally a footer. I provide an anchor and a caption in the header. Therefore, since I publish a lot Figures and Listings, I use a lot of tables of the sort shown in Figure 1 , Figure 3 , and Listing 1

Only occasionally do I need a table with multiple columns and multiple rows. The template that I will provide has examples of several different styles of multi-column, multi-row tables. This is definitely a case where it is probably easier to copy, paste, and edit than to start from scratch and build the table.

It is worth noting that if a single header or a single footer is created in a multi-column table, it normally appears above or below the first column with a width that is limited by the width of the column.

Ordinary text

All ordinary text must be in a paragraph or some other suitable element such as a table or a blockquote. You won't be able to upload your content to OpenStax if you create text in your XHTML document that isn't properly contained.

This is a serious problem because your validator may not catch it. In that case, my translator program also won't catch it and you won't get any

warning until you try to upload the CNXML file to OpenStax. If that happens, you will get upload errors from OpenStax similar to the following:

```
Please correct the following errors:
Line 2117: error: text not allowed here
Line 2118: error: text not allowed here
Line 2119: error: text not allowed here
Line 2121: error: unfinished element
Note: Edit-In-Place cannot be used while errors
persist
```

If you view the XHTML template file that I have provided in your browser, you will see that the ordinary paragraph text is displayed in the color teal *(which is sort of green)* as shown in the first and last paragraphs of Figure 2 . Without going into detail, different kinds of content in that template are displayed in different colors. *(The color information isn't transmitted to OpenStax. OpenStax displays everything in its own color scheme.)*

Figure 2. Browser text colors.

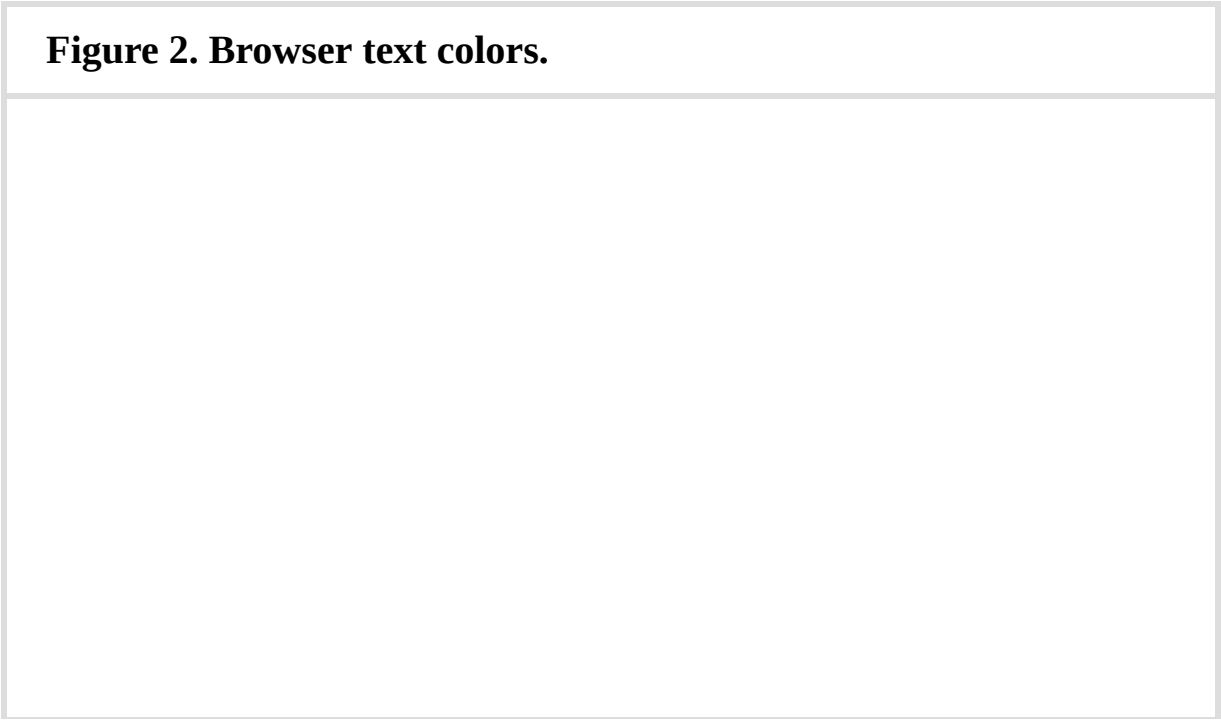
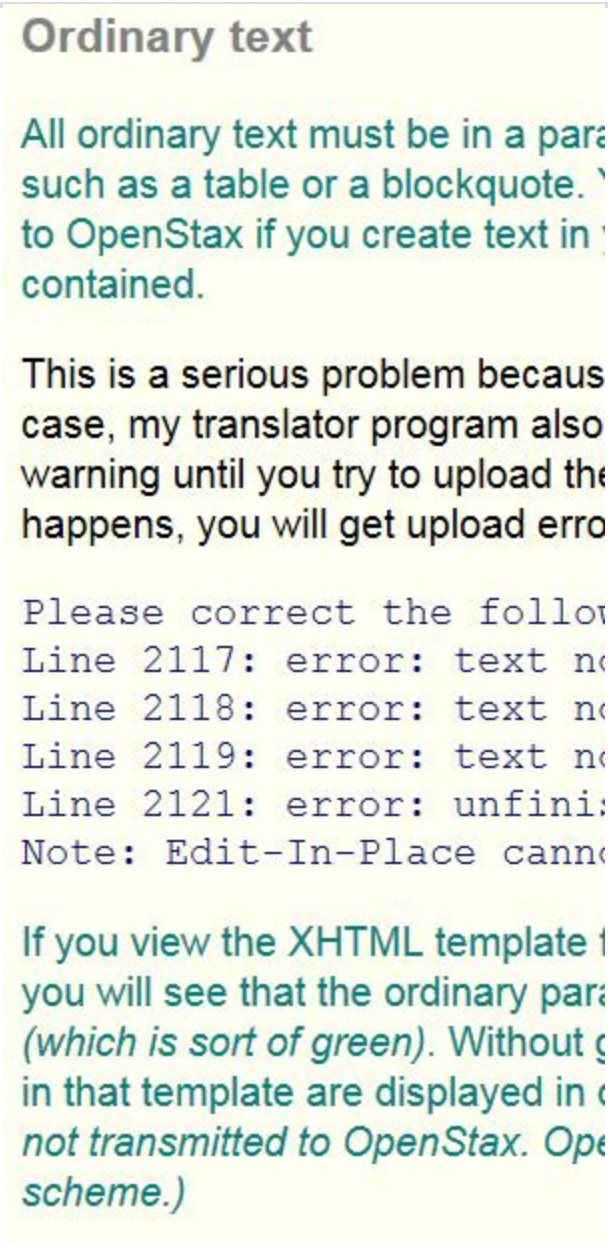


Figure 2. Browser text colors.



If you use that template and you create text that is not properly contained in a suitable element, it will be displayed in black, as shown by the second paragraph in Figure 2 , when you view the XHTML file in the browser. That usually makes it easy to spot and fix before it creates problems later on.

XHTML headers and CNXML sections

XHTML headers such as **h1** , **h2** , etc., are used as delimiters when creating **sections** in the CNXML code. CNXML sections begin and end at XHTML headers such as **h1** , **h2** , etc.

XHTML headers must be nested in a hierarchical manner. In other words, **h2** must be a child of **h1** , **h3** must be a child of **h2** , etc.

The hierarchical nature of this document is illustrated in the Table of Contents . Each item in the Table of Contents was a link to a header *(such as **h1** , **h2** , **h3** , etc.)* in the XHTML document and is a link to the beginning of a section in this CNXML document.

Extraneous entities

Once you have finished editing your XHTML file, it may contain one or more entities that can't be handled by this program. *(The most common entity is the so-called non-breaking space entity shown **below** .)*

```
 
```

Some entities *(such as those shown here)* are required while others are not allowed. Any entities that are not allowed will be identified by error messages when you run the program. As far as I know, they all begin with an ampersand and end with a semicolon as shown above .

The bad entities must be removed from the XHTML file before it can be successfully processed by this program. This is one place where your WYSIWYG editor will fail you. Probably the best way to remove entities is to switch to the code-editor window of your editor program, search out the bad entities, and delete them. Just be sure not to delete any of the good entities in the process.

Images

Free-standing images such as the one shown below are allowed by the program.

However, if like me you often need to put captions and anchors on your images, such as those shown in Figure 3 , you can put each image in a table with a **border** value of 20 and a **width** value of 0. Put the caption and the anchor in the table header, *(or you could put the caption in the optional footer)* .

Figure 3. Image in a table.

Could put a caption here.

Don't put the anchor in the footer. If you put the anchor in the header and click on a link to that anchor later when viewing the page, a typical browser

will scroll the page such that the header will be at the top of the page. Given that, putting the anchor in the footer doesn't work well for obvious reasons.

All **img** elements must be closed with the / character. Some editors *(including the free version of Microsoft Expression Web 4 which I use)* don't do that. Therefore, it is necessary for me to edit the raw XHTML code to add those / characters to the **img** elements.

Validating your XHTML file

Validate your finished XHTML file as *XHTML 1.0 Transitional* using any good XHTML validator. The free version of Microsoft Expression Web 4 has a built-in validator and it is not necessary to put a DTD declaration in your XHTML file.

Some validation software may require that you include a DTD declaration at the top of your XHTML file in order to validate the file. If that is the case with your validator, you MUST remove the DTD declaration from the XHTML file before using the file as input to this program.

A DTD declaration normally looks something like the following, beginning with **!DOCTYPE** and ending with **.dtd** and surrounded by angle brackets.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
```

If you don't want to physically remove it from your XHTML file, you can temporarily disable it by turning it into a comment as shown below:

```
<!--
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
-->
```

Don't use XHTML break tags

Break tags are used in XHTML to create a series of lines of text with no space in between. An XHTML break tag looks like this:

```
<br/>
```

To my knowledge, there is no CNXML tag that maps into an XHTML break tag. Therefore, there must not be any XHTML break tags in the temporary output file produced by an object of the class named **CNXMLFirstPass12** . The behavior of an object of that class is to replace all XHTML break tags with XHTML paragraph tags. Although this works in some cases, it can also cause "well formed" errors later downstream in other cases.

The best approach is to manually remove all XHTML break tags from your finished XHTML file before processing it with this program. In other words, DON'T USE XHTML break tags in your XHTML file unless it is absolutely necessary.

CNXML editing is not required

It should not be necessary for you to manually edit the CNXML file produced by this program either before or after uploading it to OpenStax. If there are problems with the upload of the CNXML file, or problems with the published result, it should be possible for you to make any necessary corrections at the XHTML level. Then re-run this program to create a new CNXML file for uploading. I have been doing that since 2009 and with more than 770 books and pages published on OpenStax, I have never found it necessary to edit CNXML code.

No CNXML figure objects or CNXML listing objects

Early versions of this program supported CNXML **figure** objects and CNXML **listing** objects. Although those objects sound good in theory, I

have encountered significant display problems with objects of those types, particularly when the site moved from the Legacy presentation format to the OpenStax presentation format. As a result of those problems, I had to go back and remove those objects from most of my pages. Following that, I updated the program to disallow the creation of CNXML code for **figure** objects or **listing** objects.

My approach, which works very well for me, is to put figures and listings into CNXML **table** objects and to distinguish between figures and listings in the captions, which I place in the table header. *(See Figure 3 and Listing 1 for example.)*

Run the program

This section explains how to run the program and how to publish your CNXML content on OpenStax.

Running the program

This program requires access to the following three compiled Java class files plus a valid XHTML file for input

- CNXMLprep12.class
- CNXMLFirstPass12.class
- CNXMLSecondPass12.class

Click here to download a zip file named **CNXMLprep12.zip** containing those three class files. Click here to download a zip file named **CNXMLtemplate01.zip** containing a valid XHTML template file named **CNXMLtemplate01.htm** that illustrates many of the capabilities of this program. The zip file also contains several image files associated with the template file.

To run this program in general, extract the contents of the zip file named **CNXMLprep12.zip** into an empty folder, copy your validated XHTML file

into that folder, and execute the following command at the command prompt in that folder:

```
java CNXMLprep12 InputFile OutputFile
```

where **InputFile** is the name of your validated XHTML file and **OutputFile** is the name of the CNXML file that the program will produce.

This program was developed using the Java SE Development Kit 8 . In order to run this program on your computer, you must have the Java SE Runtime Environment 8 (JRE) or a later version of the JRE installed on your computer. The JRE is already installed on many computers and it may already be installed on yours *(but it may be an earlier version that requires updating)* . If it is not installed, or it is not up to date, there are many web pages that provide instructions for downloading and installing Java. One of those web pages is located here .

To run this program with the template file named **CNXMLtemplate01.htm** , extract the contents of both zip files into an empty folder. Then execute the following command at the command prompt in that folder:

```
java CNXMLprep12 CNXMLtemplate01.htm
CNXMLtemplate01.cnxml
```

Two new files should appear in the folder:

- CNXMLtemp12.html
- CNXMLtemplate01.cnxml

You can ignore the temp file. It is of no further use.

The file named **CNXMLtemplate01.cnxml** is the output CNXML file produced by the program.

Upload and publish your new page

Here is an abbreviated description of the steps for uploading and publishing your new page. If you need more detailed instructions, see Step-by-Step Guide to OpenStax Publishing .

1. Click here to create an OpenStax account and sign in to your new account. *(See helpful hints below)*

2. Create a new **workgroup** or use the workgroup named **Personal Workspace** to create a new **module** .

> **Historical context** : In the days when the website was known as **Connexions** , before it became known as **OpenStax** , individual CNXML files were known as **Modules** and a collection of such files was known as a **Collection** . With the advent of the **OpenStax** name, individual CNXML files became known as **Pages** , and collections of such files became known as **Books** . Much of the old terminology still appears on the website, particularly in those areas of the site associated with creating and publishing content. Therefore, you will be creating a **Module** , which will be known as a **Page** once it is published and viewed in the **OpenStax** format. However, if you view it in the **Legacy** format *(link on the upper-right corner of the **Page** screen)* , it will still be known as a **Module** .

3. Enter the Metadata for the new module *(title, keywords, summary, etc.)*

4. Upload the file named **CNXMLtemplate01.cnxml** to OpenStax as type ***Plain CNXML*** .

5. Put the five image files that you downloaded earlier *(**0135ex02.jpg** through **0135ex06.jpg**)* in a zip file and upload it to OpenStax as type ***Zip File*** .

> **Important Note** : The XHTML code must be written so as to access image files and other resource files from within the same

> folder as the validated XHTML file. References to resource files in other folders are not allowed in the XHTML code.

6. Finally, follow the OpenStax instructions to publish your new content.

Helpful hints regarding OpenStax authoring

Once you have signed in to OpenStax, go to Authoring Content . That page provides links to several other pages containing helpful hints regarding the authoring of content on OpenStax. The page titled Create a Module might be particularly helpful in describing the steps involved in creating and publishing a module. Just remember, that you already have the CNXML code for your module. You simply need to import and publish the module. *(You will be importing CNXML instead of Microsoft Word.)*

You will also find links to other useful authoring pages on your MyCnx page.

The utf8 character set

The XHTML input file must be coded as **charset=utf8** . Be careful with this requirement because some XHTML editors create text using other character sets. The safest approach is to start with my XHTML template file that contains a **meta** line in the **head** section that looks something like the following:

```
<...charset=utf-8...>
```

Then edit that template file to meet the needs of your page.

Why not create and upload Microsoft Word documents to OpenStax ?

The OpenStax documentation indicates that the following file types can be imported:

- Microsoft Word
- OpenOffice Writer
- LaTex
- Plain CNXML
- Zip File

(If you use my XHTML-to-CNXML translator, you will be importing "Plain CNXML" and "Zip File" into OpenStax.)

Now for the primary question posed by this section -- to begin with, when you select Microsoft Word as the import type, you are immediately confronted with the following disclaimer:

Note:
Word documents can be extremely varied. We have attempted to handle many common cases, with the goal of extracting the text from your document so that ***you can mark it up in CNXML*** without having to retype the content. Some cautions:

- Images: Many images will be imported without trouble. Images contained in figures or tables, created directly in Word, or embedded in the document through links may not work correctly.
- Styling Tips: If you use CNXML-specific styles from the Word template to create your Word document, you will get much more faithful conversion for elements like terms, citations, code, and others. See the styling information in our full instructions, below.
- Use Headings to Get Sections: Even without CNXML-specific styling, you can make sure that your section organization is preserved during import by using the standard header styles (Heading 1, Heading 2, etc.) in your original document.
- Full instructions on using Word with OpenStax-CNX

That is hardly a confidence builder. If you follow the link to the Full instructions , you will find the following at the beginning of the page.

Note:
Overview
One of the easiest paths for populating a module is the Word/OOo importer, which converts a *.doc document to CNXML. The importer overwrites any existing CNXML, so it is most useful as an ***initial import. We suggest using the online Edit-in-Place feature to make any further edits*** ; any re-import of a Word or OpenOffice document will erase any other changes you have made using Edit-in-Place or the Full Source Editor.
Who should use the Word/OOo importer?
You should use the Word/OOo importer if you already have a Word or OpenOffice document saved that you wish to publish to the repository...

It would appear from the above that the primary purpose of the Word importer is to provide a "starting point" CNXML document to be further edited online in CNXML code.

Note: *(In contrast, if you create your document in WYSIWYG XHTML and use my XHTML-to-CNXML translator to prepare it for upload to OpenStax as type Plain CNXML, you never have to touch or even think about CNXML code. You can handle creation as well as ongoing maintenance at the WYSIWYG XHTML level.)*

Actual results : I opened this XHTML file in Microsoft Word and then saved the file both as a **doc** file and a **docx** file. Then I tried to import both of those files into OpenStax as type Microsoft Word. In both cases, I received an error message that read " **Could not import file. Could not convert file** ."

There was no indication as to the cause of the problem nor was there any suggestion as to how to resolve the problem. Therefore, in my case, trying to import a Microsoft Word document into OpenStax has proven to be a dead end street.

That is my reason for not creating and uploading Microsoft Word documents to OpenStax?"

Summary

If you can create a valid XHTML file using a WYSIWYG editor such as the free version of Microsoft Expression Web 4, you can easily create and publish content on OpenStax using my free XHTML-to-CNXML translator program. This page explains how to use my translator program to create and publish your first page on OpenStax.

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Wysiwyg0100 Getting Started
- File: Wysiwyg0100.htm
- Published: 03/18/16

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Hosting PDF on OpenStax
This page describes a process by which you can easily host your PDF content on OpenStax.

Revised: Thu Mar 24 15:10:14 CDT 2016

Note:
This page is part of a Book titled OpenStax Publishing with a WYSIWYG Editor.

This page is designed specifically to host a PDF file.

Click here to view the PDF file .

Click here to download a zip file named template.zip.

-end-

Step-by-Step Guide to OpenStax Publishing
The purpose of this Page is to guide you through the uploading and publishing process, step-by-step, using screen shots to illustrate the steps involved.

> Revised: Thu Mar 24 14:56:00 CDT 2016
>
> **Note:**
> This page is part of a Book titled OpenStax Publishing with a WYSIWYG Editor.

Table of contents

Preface

The purpose of this Book is to provide the tools needed for persons who have never published on OpenStax to create, upload, and publish their content on OpenStax. Previous pages have showed you how use a WYSIWYG editor to create content and then how to translate that content into CNXML for uploading and publishing on OpenStax.

The purpose of this **Page** is to guide you through the uploading and publishing process, step-by-step, using screen shots to illustrate the steps involved. Instructions provided on this **Page** assume that you have already created your CNXML file, possibly but not necessarily using one of the following two approaches:

- Wysiwyg0100 Getting Started
- Hosting PDF on OpenStax

Viewing tip

I recommend that you open another copy of this **module** in a separate browser window and use the following links to easily find and view the Figures while you are reading about them.

Figures

Background information

At one point in time, the **OpenStax** website was known as **Connexions** . In those days, individual CNXML documents were known as **modules** and a collection or grouping of such documents was known as a **collection** . With the advent of the **OpenStax** name, individual CNXML documents became known as **Pages** , and collections of such documents became known as **Books** . However, much of the old terminology still appears on the website, particularly in those areas of the site associated with creating and publishing content.

Therefore, a **module** is the same as a **Page** and a **collection** is the same as a **Book** . Those terms will be used somewhat interchangeably on this **Page** .

Discussion

Create an account

The first step in publishing on OpenStax is to create a free account if you don't already have one. *(An account is not required to view content on OpenStax, but an account is required to upload and publish content.)*

If you already have an account, skip ahead to Log into your account . Otherwise, the best place to start when creating an account is here . When you select that link, you should see something similar to the page shown in Figure 1 .

Figure 1. Starting page for authors on OpenStax.

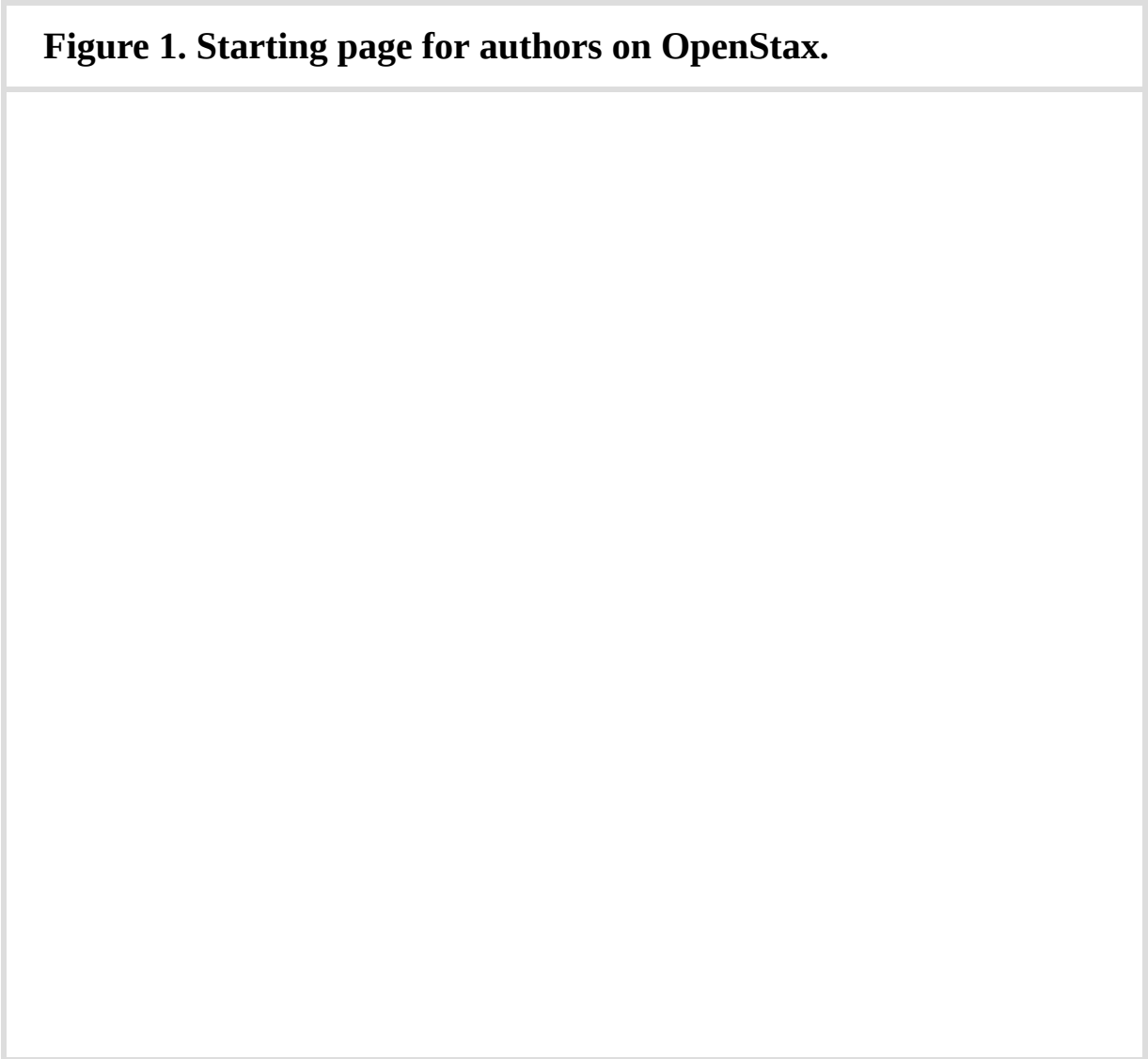
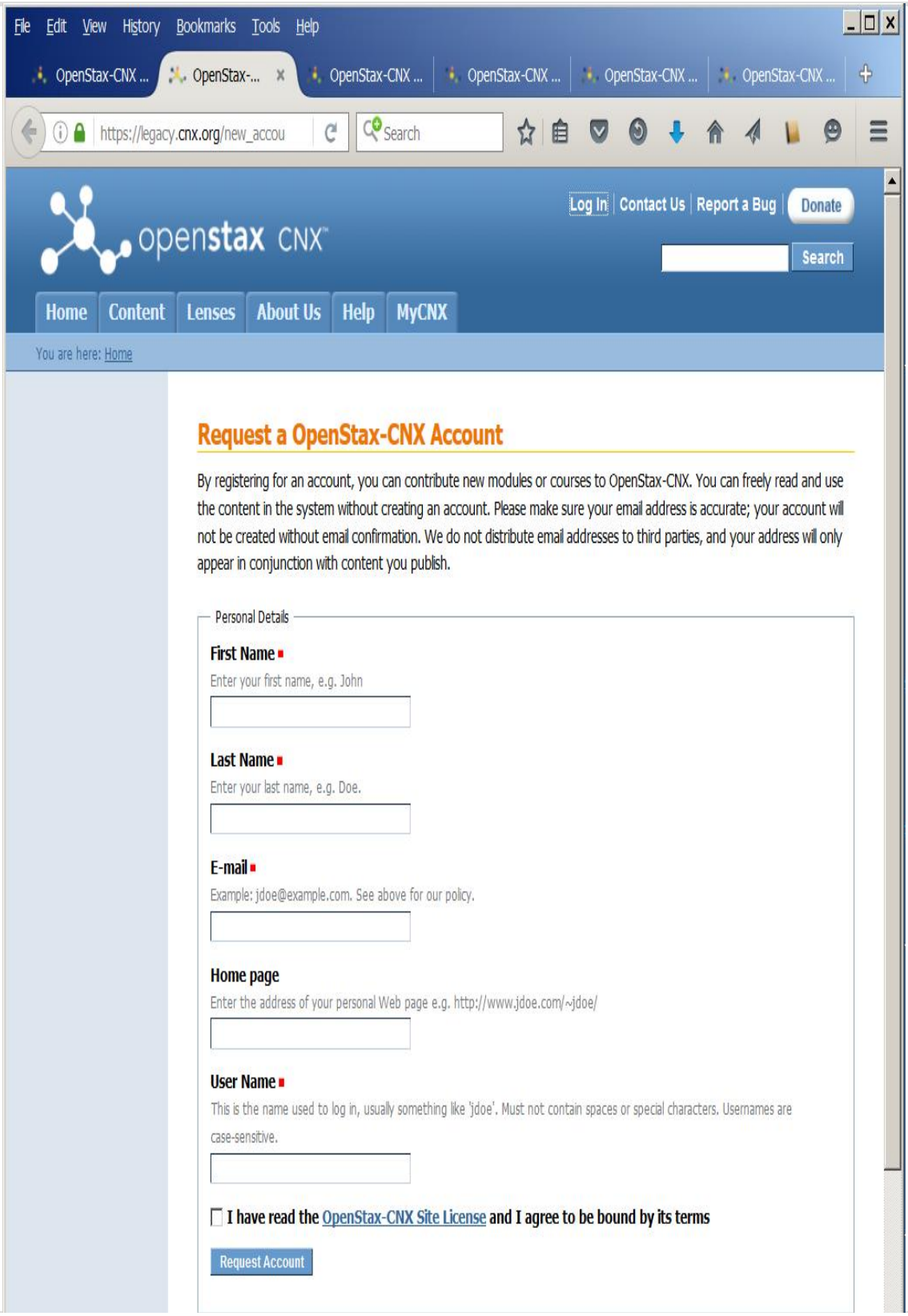


Figure 1. Starting page for authors on OpenStax.

Take time to look around when you get there. Then select the link titled Get an account in the upper-right of the page. That should take you to a page that looks something like Figure 2 .

Figure 2. Request a OpenStax-CNX Account.

Figure 2. Request a OpenStax-CNX Account.



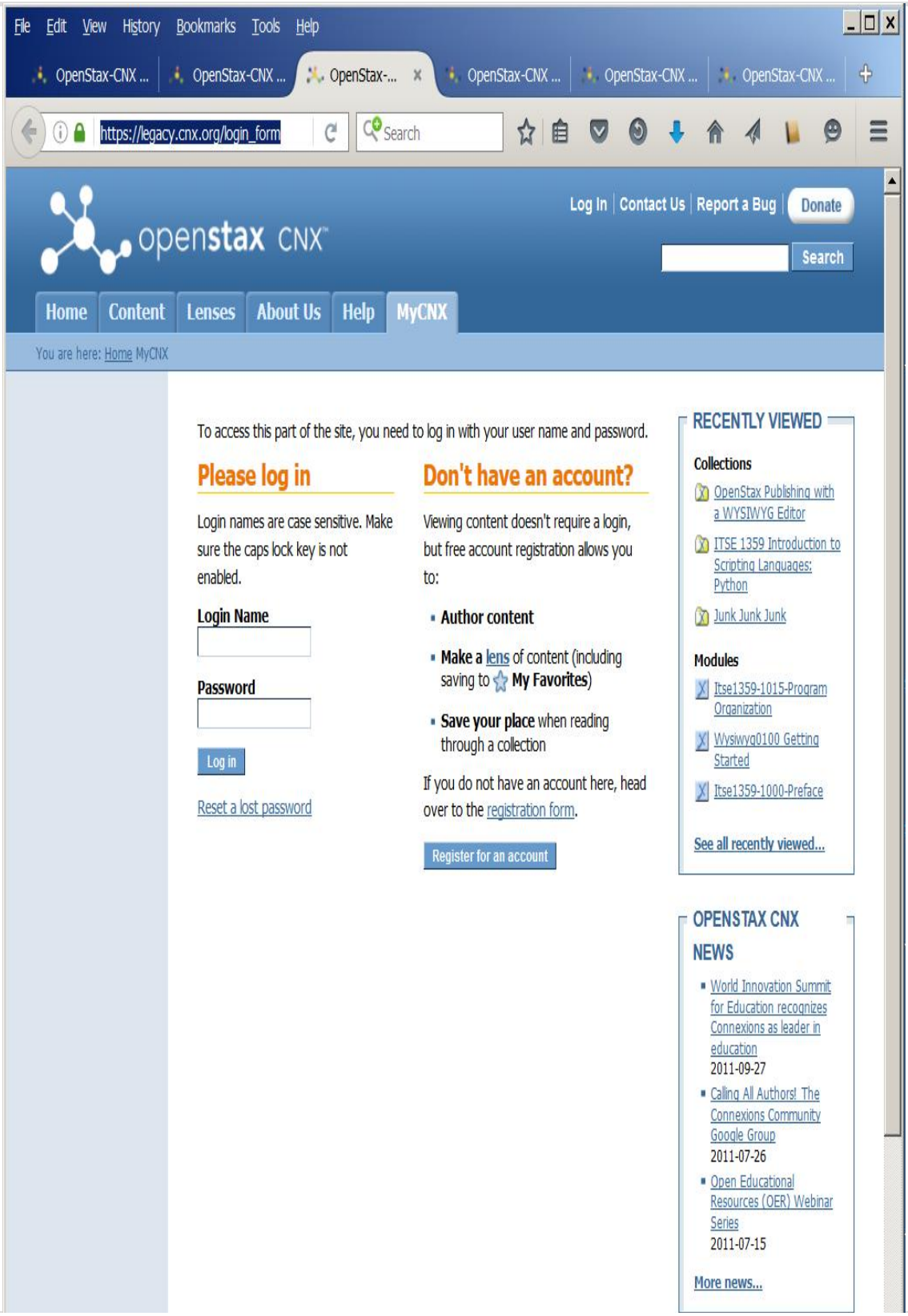
Go ahead, follow the instructions and get your account. Once you have your account, log out and log back in to make certain that your password works, etc.

Log into your account

You will find the login page at https://legacy.cnx.org/login_form . It should look something like the page shown in Figure 3 although it won't have some of the material that you see there the first time you log in.

Figure 3. The login page.

Figure 3. The login page.



Go ahead and log in. When you do you should land on a page that looks something like Figure 4 .

Figure 4. The login landing page.

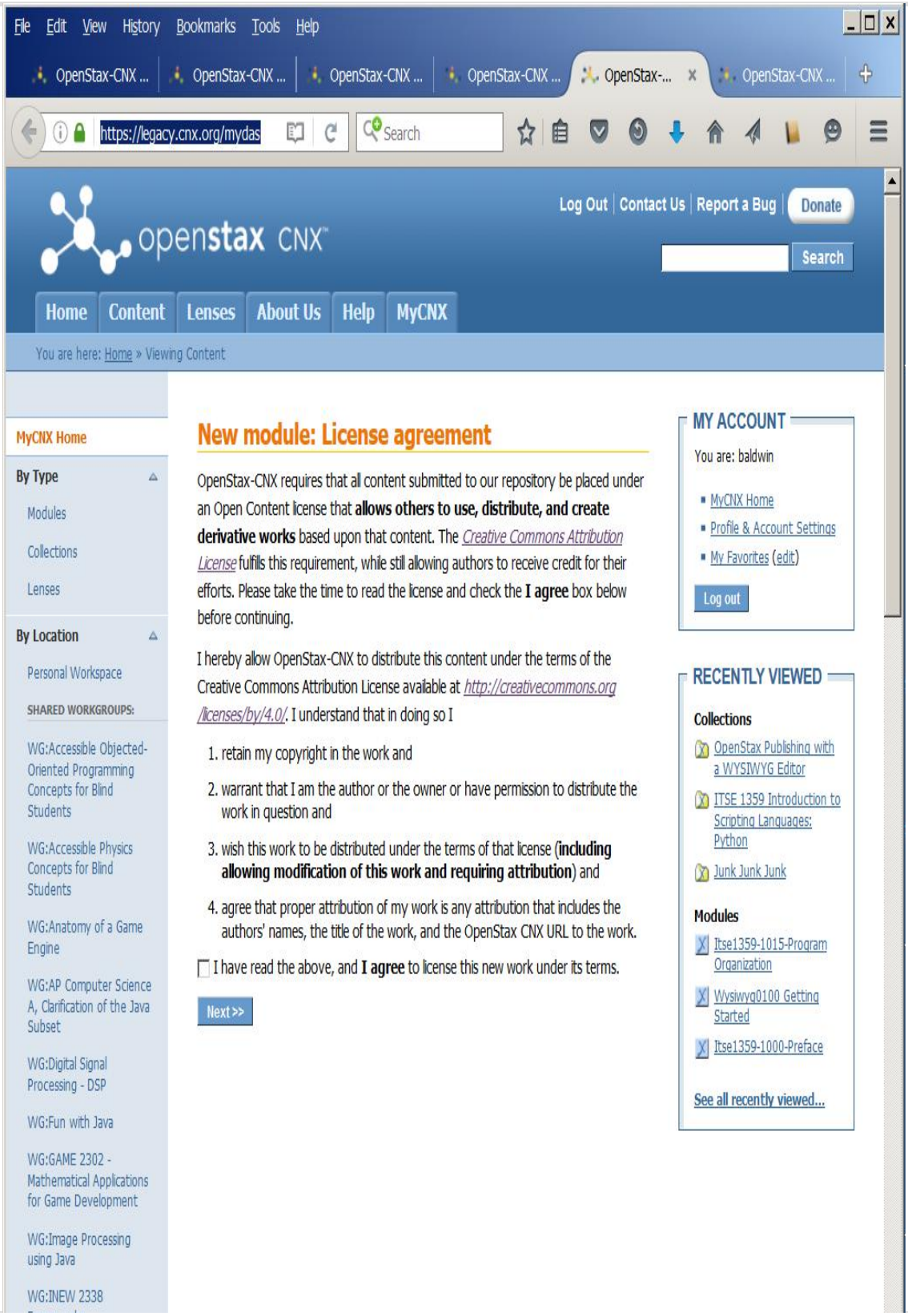
Figure 4. The login landing page.

Create a new module

Select the link to **Create a new module** that appears under the heading titled **Create and edit content** . When you select that link, you should land on a page that looks something like that shown in Figure 5 .

Figure 5. New module: License Agreement.

Figure 5. New module: License Agreement.

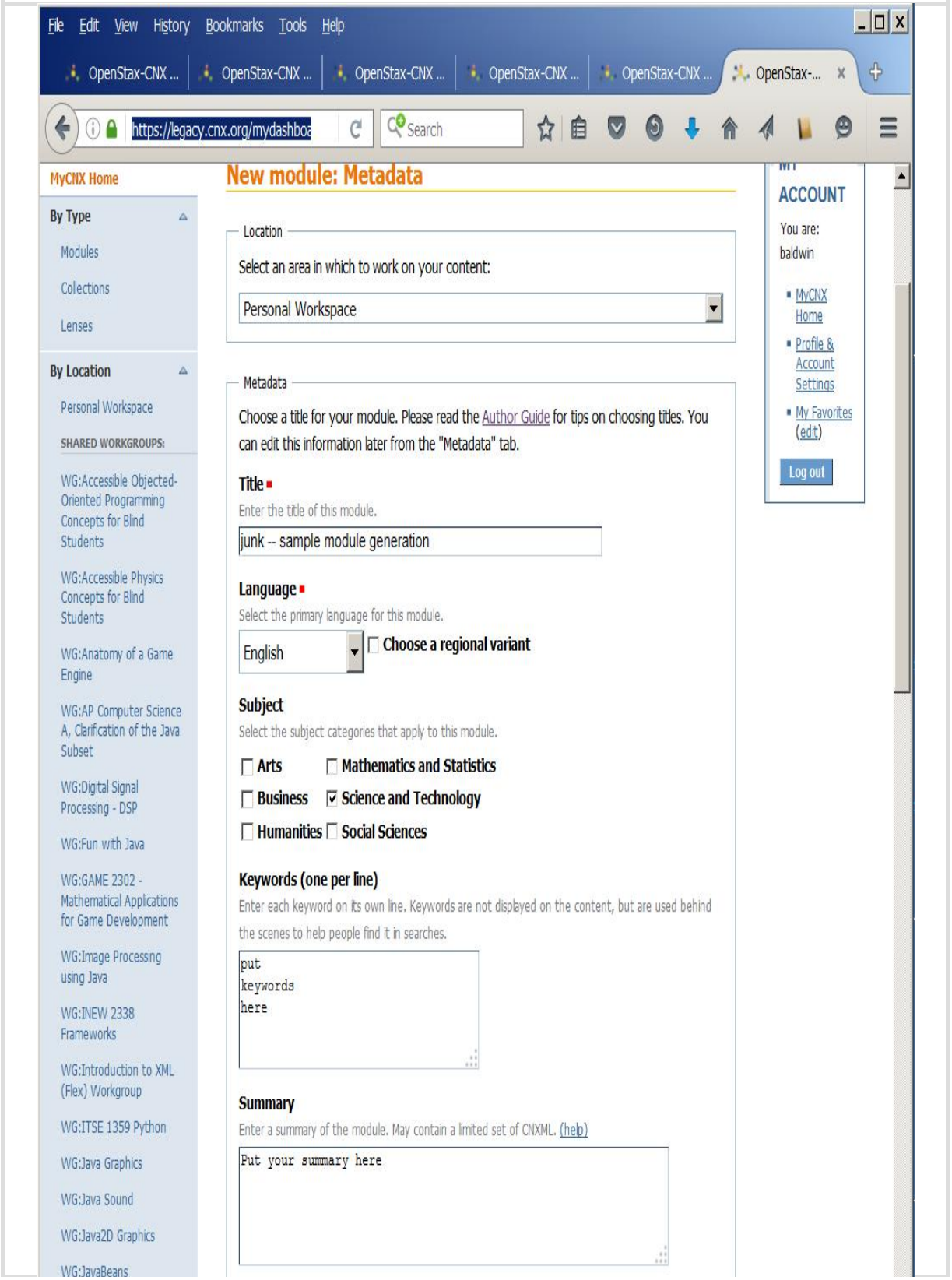


Accept the license

Read the license, check the box, and click the **Next** button. When you do that, you should land on the **New module: Metadata** page shown in Figure 6 .

Figure 6. New module: Metadata.

Figure 6. New module: Metadata.



A word about work spaces

Note first that your new **module** is being created in your **Personal Workspace** , which is the default workspace for new modules. Later on, you can create other work spaces, which is a useful mechanism for separating your work into categories. For example, if you decide to create one **Book** on cats and another **Book** on dogs, it might make sense to separate the modules belonging to each **Book** into its own appropriately named work space. *(The list of items that you see in the leftmost column in Figure 6 is a partial list of my work spaces, also called work groups.)*

For the time being, we will stay with the default **Personal Workspace** .

Enter the metadata

As you can see from the form in Figure 6 , the metadata consists of

- Title
- Language
- Subject
- Keywords
- Summary

The **title** is how your **module** is identified on OpenStax. For example, see my **module** titled Java3000: The Guzdial-Ericson Multimedia Class Library , which is included as a **Page** in my **Book** titled Object-Oriented Programming (OOP) with Java . If you do an advanced search for an author named Richard Baldwin, the search results showing **Books** and **Pages** will be reported by title.

The **language** item in the metadata is self-explanatory.

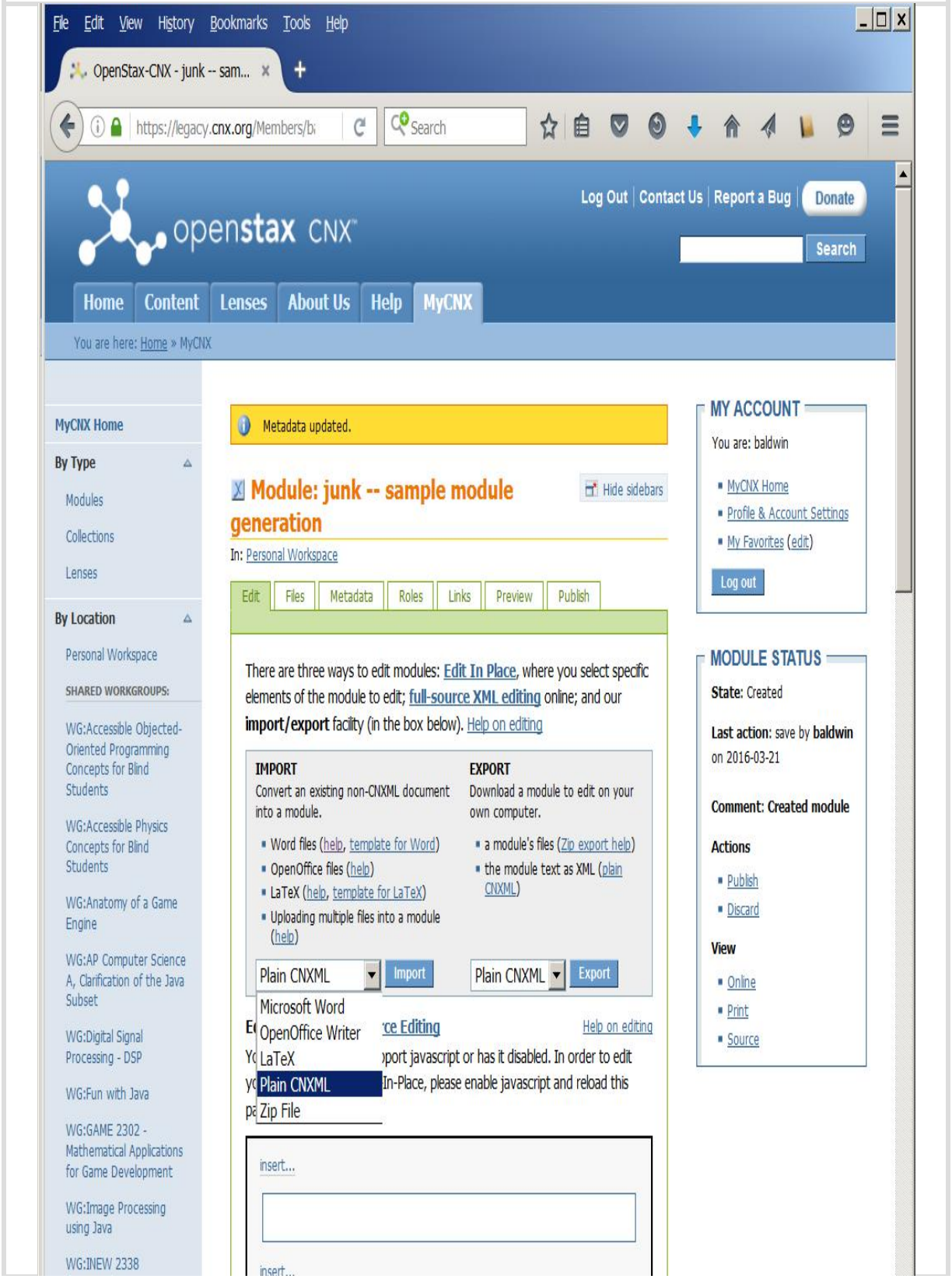
The **keywords** come into play if you do a search by keyword. For example if you do an advanced search for the keyword **polymorphism** , the search results will show the **Books** and **Pages** for which the author has declared **polymorphism** as a keyword in the metadata for the modules.

The **summary** is just that; a paragraph or two that provides a summary of the **module** . Summaries show up in a variety of locations including near the top when a **Page** is displayed in a browser. For example, go to Java3000: The Guzdial-Ericson Multimedia Class Library and click on the word **Summary** near the top of the page.

Go ahead and enter the metadata for your new **module** and then click the **Next** button. When you do, you should land on a page similar to that shown in Figure 7 .

Figure 7. Import CNXML and zip files.

Figure 7. Import CNXML and zip files.



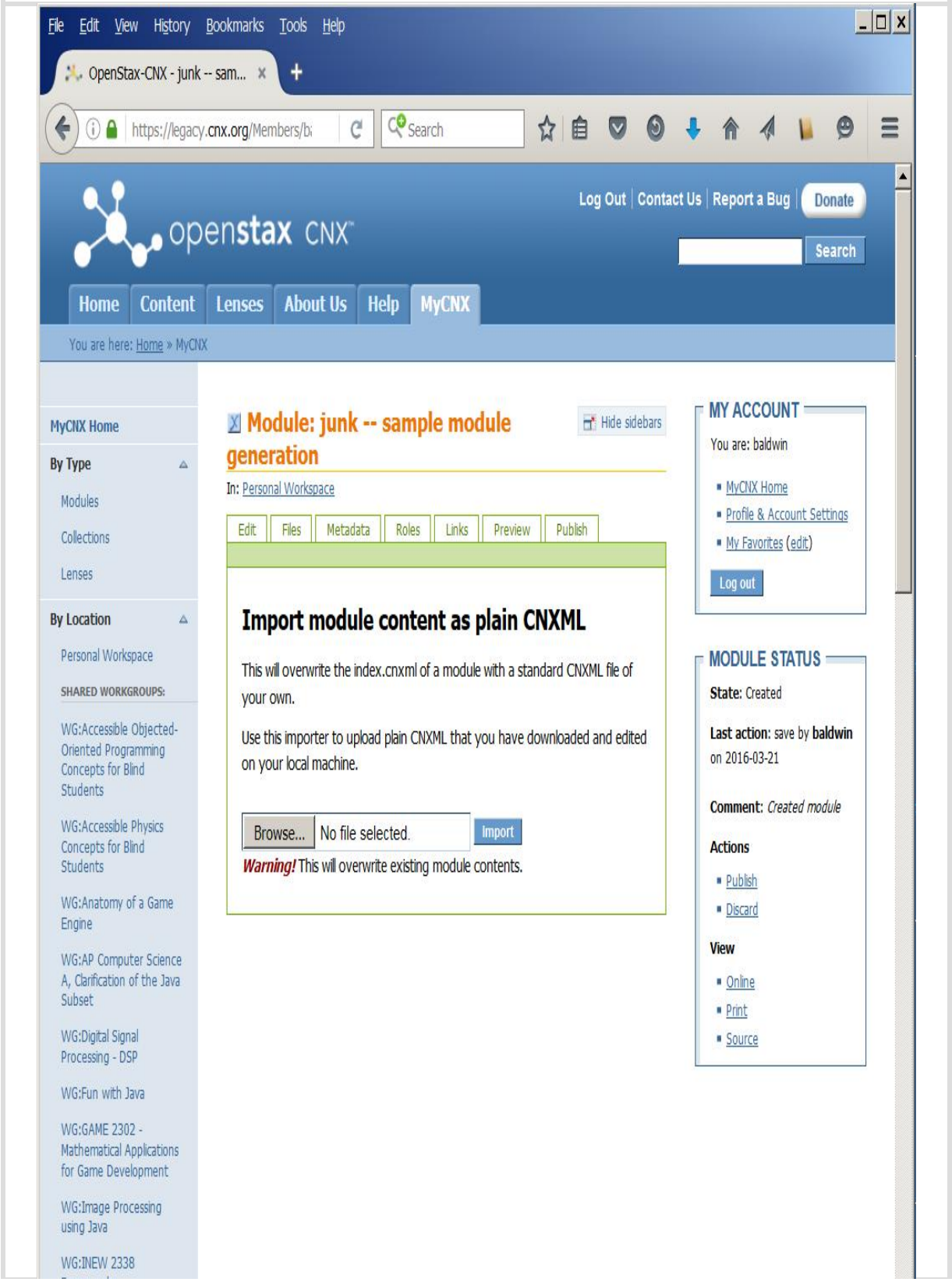
Import CNXML file

Figure 7 shows the page from which you import your CNXML file and your optional zip file. There is a pull-down list about two-thirds of the way down the page just to the left of center and immediately to the left of a button labeled **Import** . To import a CNXML file, pull the list down, select **Plain CNXML** , and click the **Import** button. To import a zip file, pull the list down, select **Zip File** , and click the **Import** button.

The page will then change to that shown in Figure 8 with a **Browse...** button.

Figure 8. Page with a Browse... button.

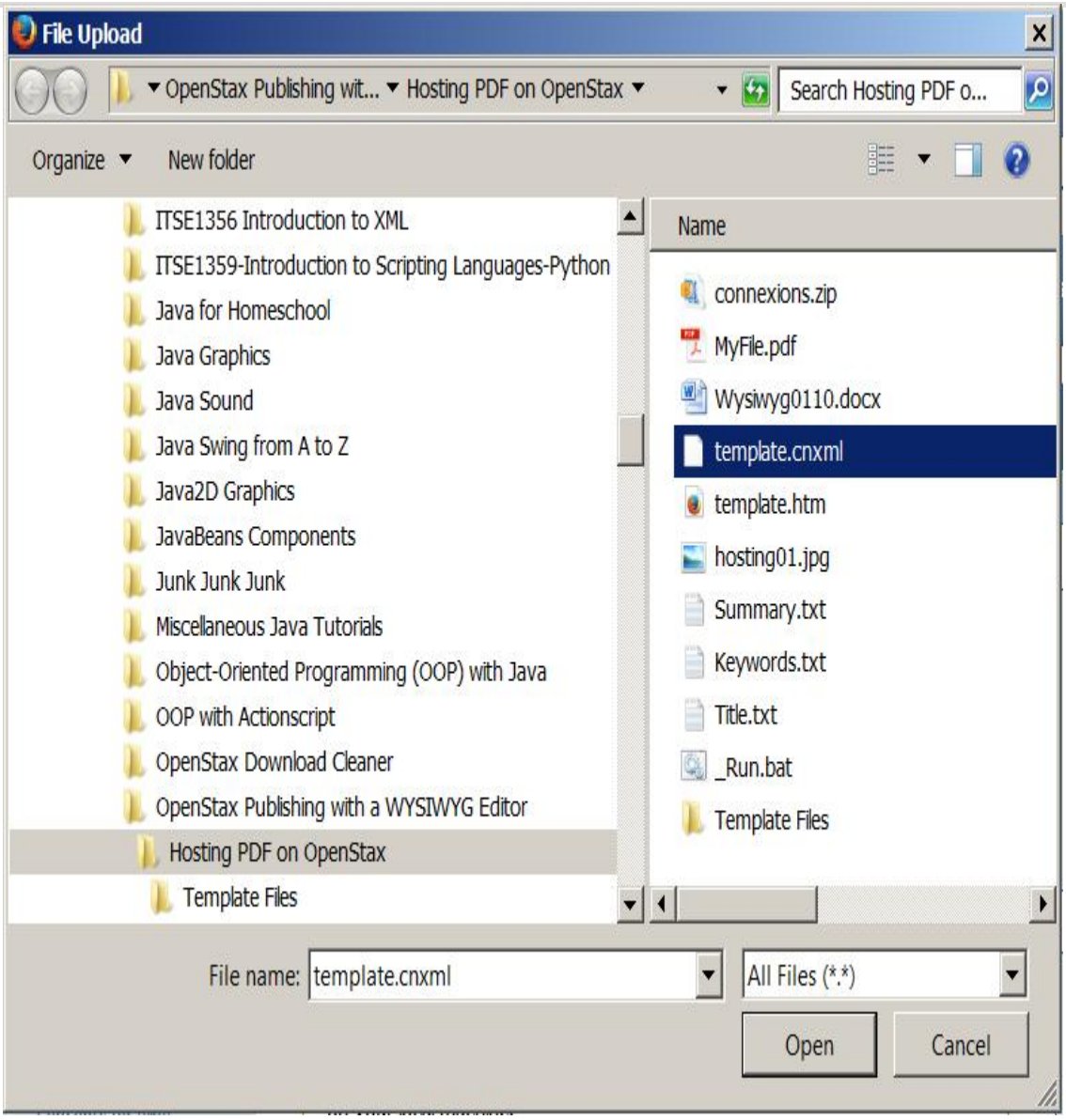
Figure 8. Page with a Browse... button.



Click the **Browse** button and a typical Browse dialog similar to that shown in Figure 9 will appear.

Figure 9. Typical Browse dialog box.

Figure 9. Typical Browse dialog box.



Browse to the location of your CNXML file, select your CNXML file, and click the **Open** button. That will place the name of your CNXML file in the box to the left of the **Import** button. You will need to click the **Import** button one more time to cause the actual import to take place.

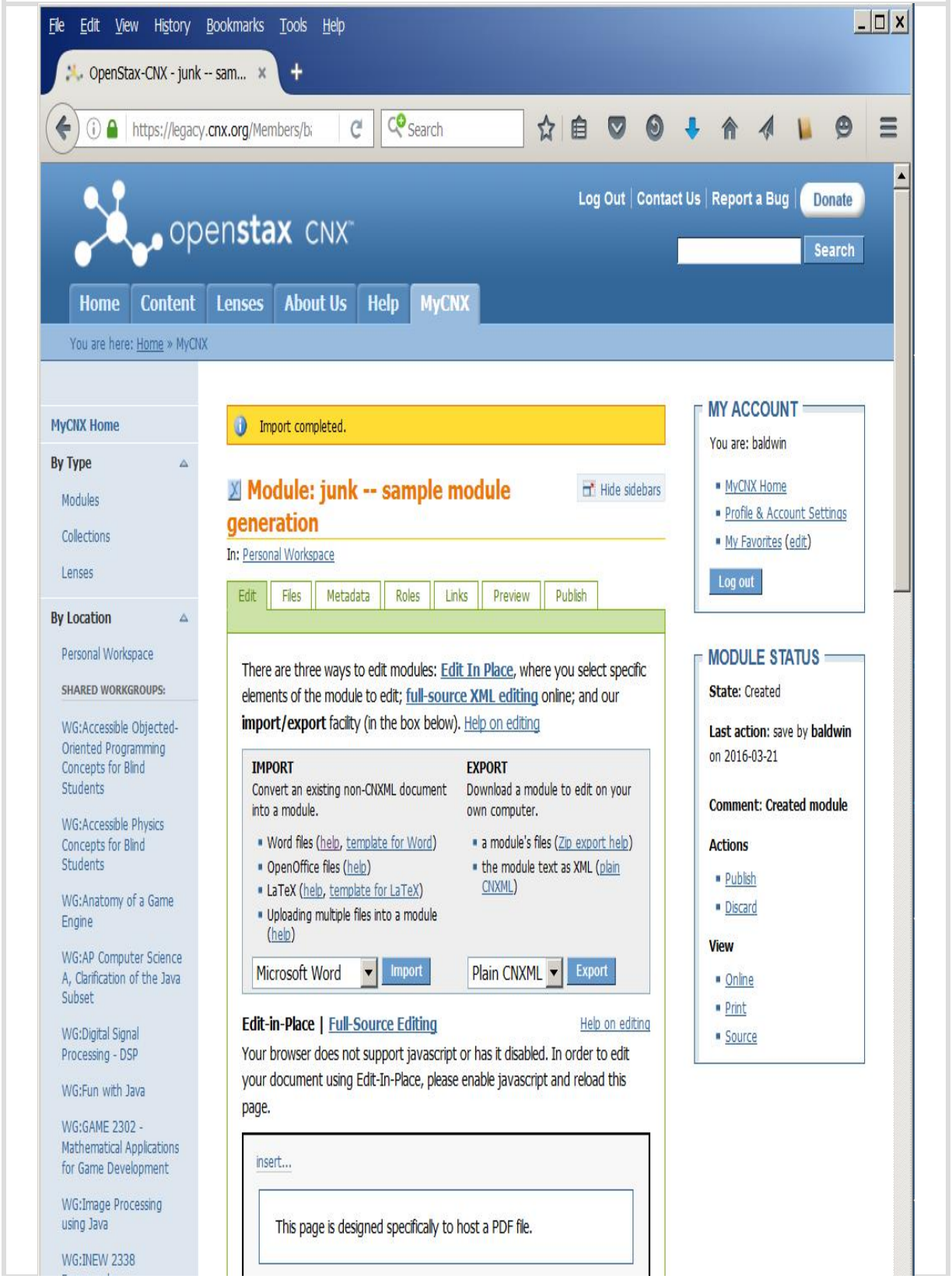
This is where CNXML compatibility errors will appear if there are any. For example, if your original XHTML file contained an anchor, which in turn

contained a comma, that will cause an error that will be identified at this point in the process. *(Hopefully you learned not to do that when you studied the **Page** titled Wysiwyg0100 Getting Started .)*

Also, hopefully there won't be any import errors and you will see a screen similar to that shown in Figure 10 . Note the yellow banner at the top with the good news: **Import complete**

Figure 10. Import complete.

Figure 10. Import complete.



Import zip file

If your **module** includes a zip file in addition to the CNXML file, repeat the import process to import the zip file.

Preview your module

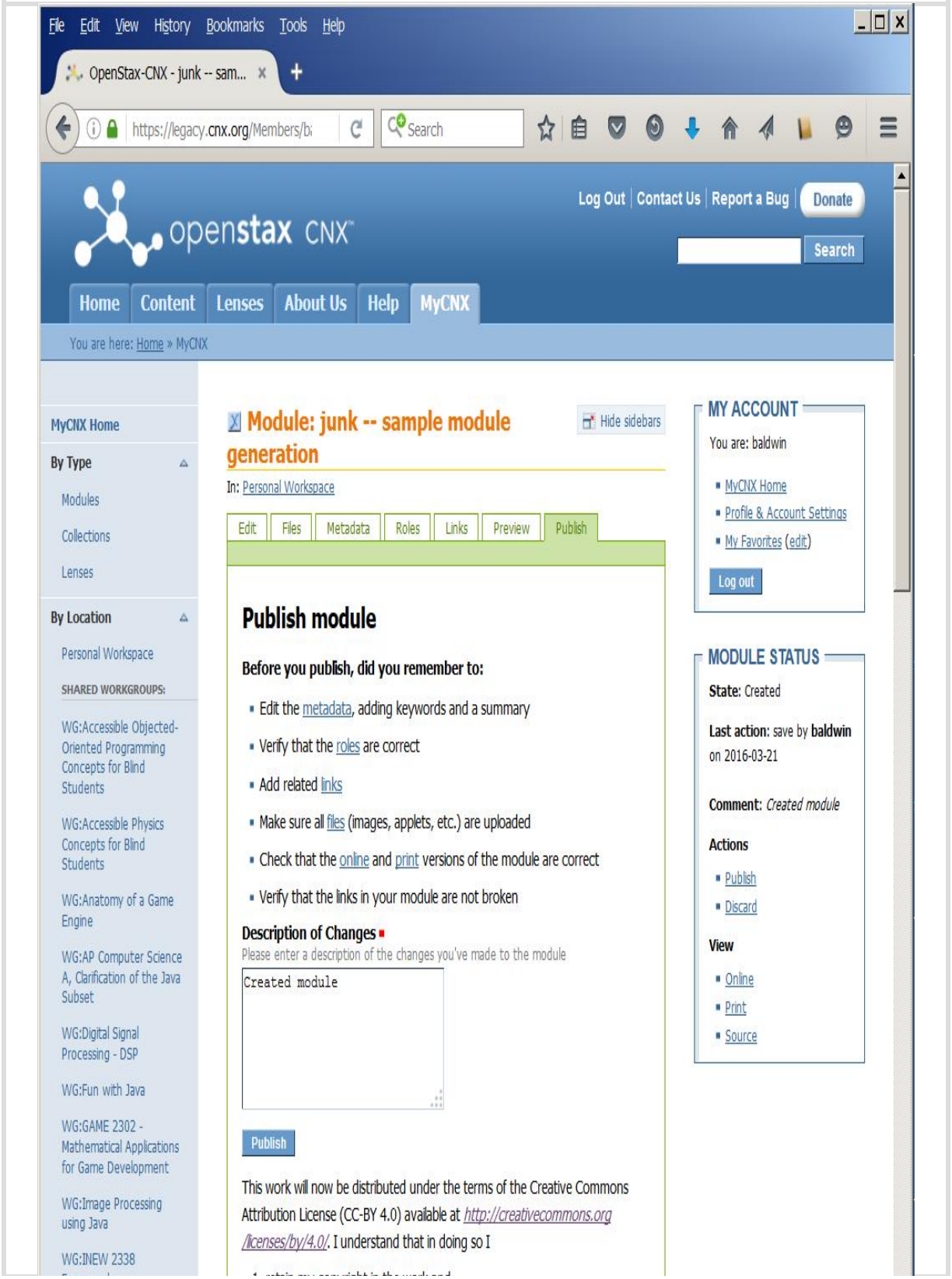
Congratulations, you have just created a **module** . If you would like to preview it before you publish it, select the **Online** link under **View** shown on the right in Figure 10 . That will give you a preview of your **module** in Legacy format. *(As of March 2016, it is not possible to preview the **module** in the more modern OpenStax format but this may change at some point in the future.)*

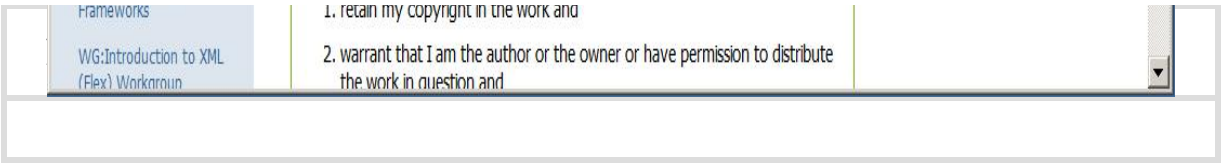
Publish your module

To publish your **module** , select the **Publish** tab at the top of Figure 10 or the **Publish** link under **Actions** on the right in Figure 10 . In either case, you should land on a page similar to that shown in Figure 11 .

Figure 11. Publish module.

Figure 11. Publish module.





OpenStax uses a version control system that makes every version of every **module** ever published available for viewing. Thus, when you publish or republish a **module** , the system wants you to provide a description of the change that led to this version of the **module** . When you first publish a **module** , the description defaults to "Created module" but you can change that if you want to.

Click the **Publish** button to cause your new **module** to be published. The first time you publish a **module** , this will cause you to land on a page that explains something to the effect that once you publish a **module** , you cannot retract it. That page asks you to confirm that you really do want to publish the **module** . Click the button labeled **Yes publish** on that page to continue. *(If you update and re-publish a **module** , you are not asked for that confirmation.)*

After you click the **Publish** button in Figure 11 , and confirm that you really do want to publish the **module** for the first time, you should land on a page similar to that shown in Figure 12 .

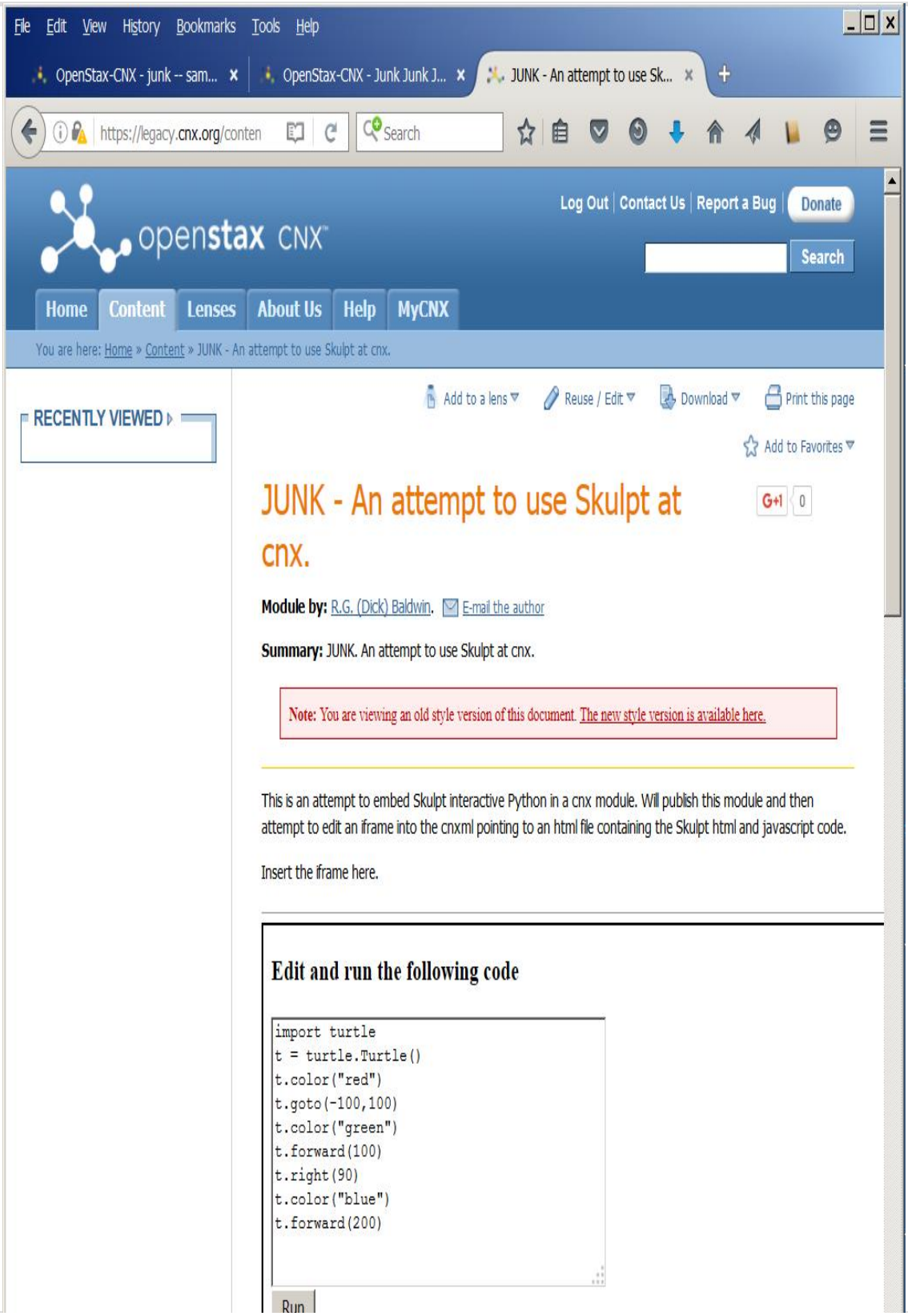
Figure 12. Item published.

Figure 12. Item published.

This page confirms that the **module** has been published and provides some information about the **module** . This page also provides a link that reads **view published version online** near the center of the page. Select that link and you should see something similar to Figure 13 , which is the legacy view of a **module** . Of course, yours will look different because it contains different content.

Figure 13. Legacy view of a module.

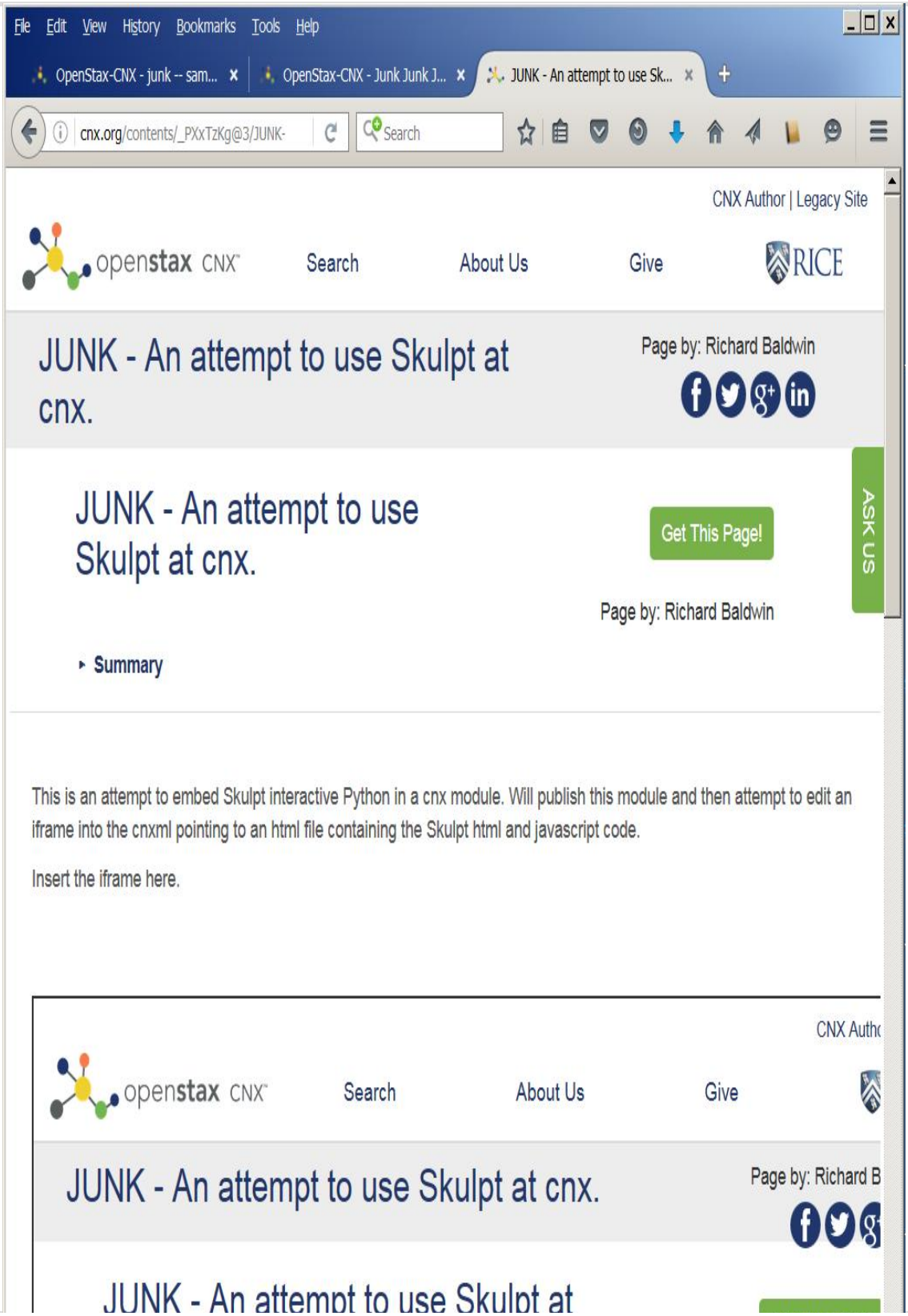
Figure 13. Legacy view of a module.



Near the center of that page is a link that reads **The new style version is available here** . Select that link and you should see something similar to Figure 14 , which is the OpenStax **Page** view of a **module** .

Figure 14. OpenStax view of a module.

Figure 14. OpenStax view of a module.



Create a collection

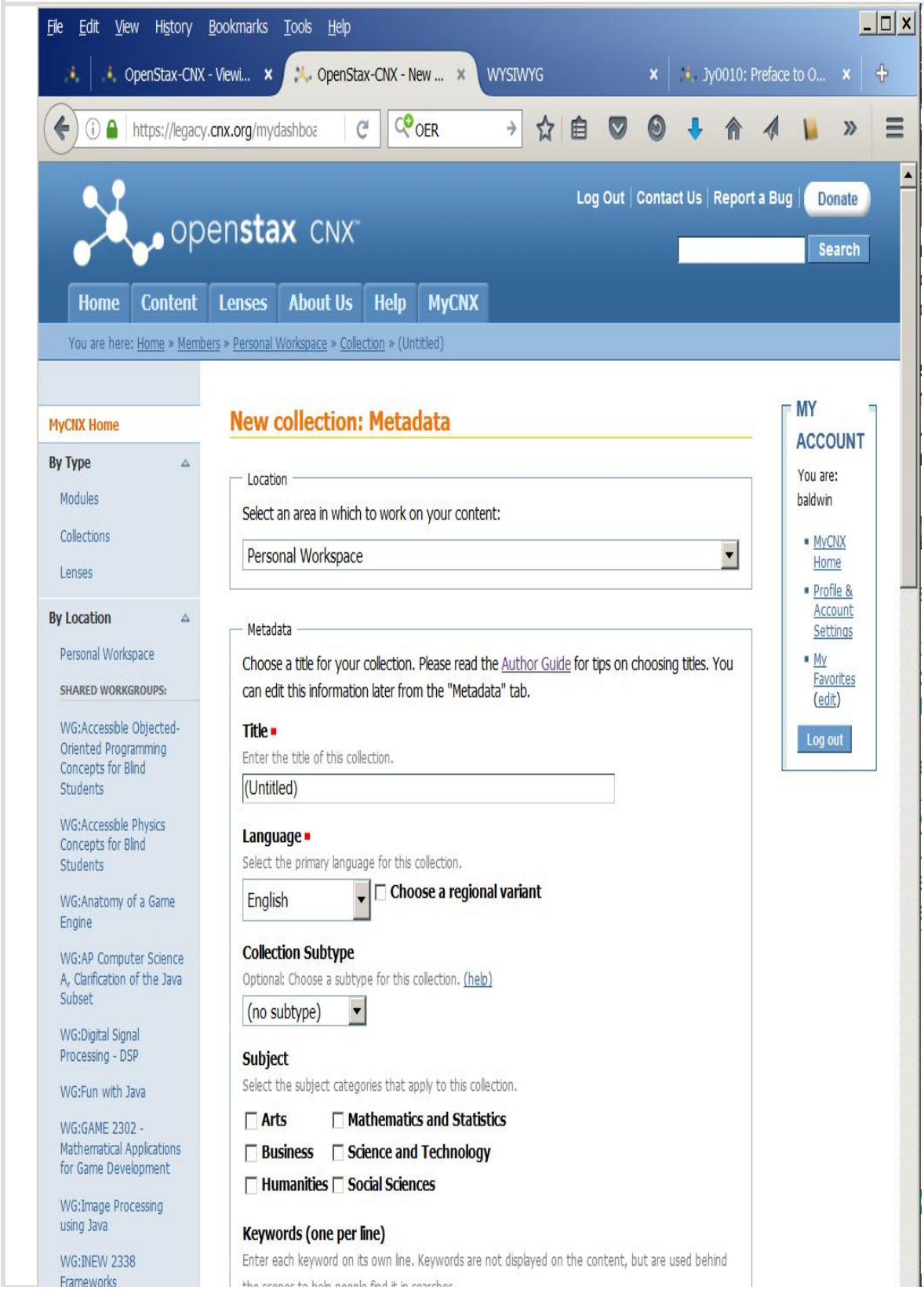
An OpenStax **module** or **Page** is a stand-alone independent document. It has its own identity independent of all other modules and collections.

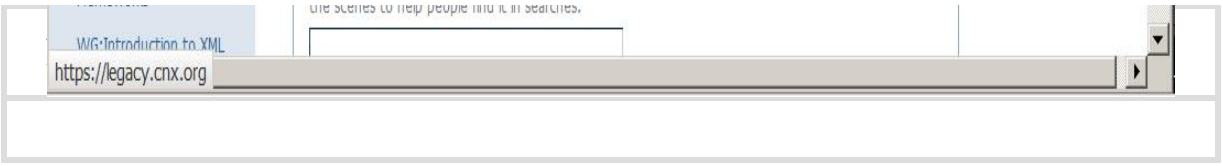
Sometimes it is useful to create a **collection** and to group two or more modules under the umbrella of that **collection** . For example, my **Book** titled Object-Oriented Programming (OOP) with Java groups many modules into a **collection** , which is actually structured into sub-collections. However, such grouping has no impact on the independence of the modules. Any **module** may or may not be grouped with other modules to form a **collection** or a **Book** . Furthermore, a **module** may be included in two or more groups that comprise independent collections.

I am not going to guide you through the creation and population of a **collection** in a step-by-step manner. However, I will hit some of the high spots. Go back to Figure 4 as the starting point. Instead of selecting the link to **Create a new module** , select the link to **Create a new collection** . Once again you will have to accept the license and after you have done that, you should land on a page very similar to that shown in Figure 15 .

Figure 15. New collection: Metadata.

Figure 15. New collection: Metadata.





You should recognize Figure 15 as being very similar to Figure 5 in which you enter the metadata for a **module** .

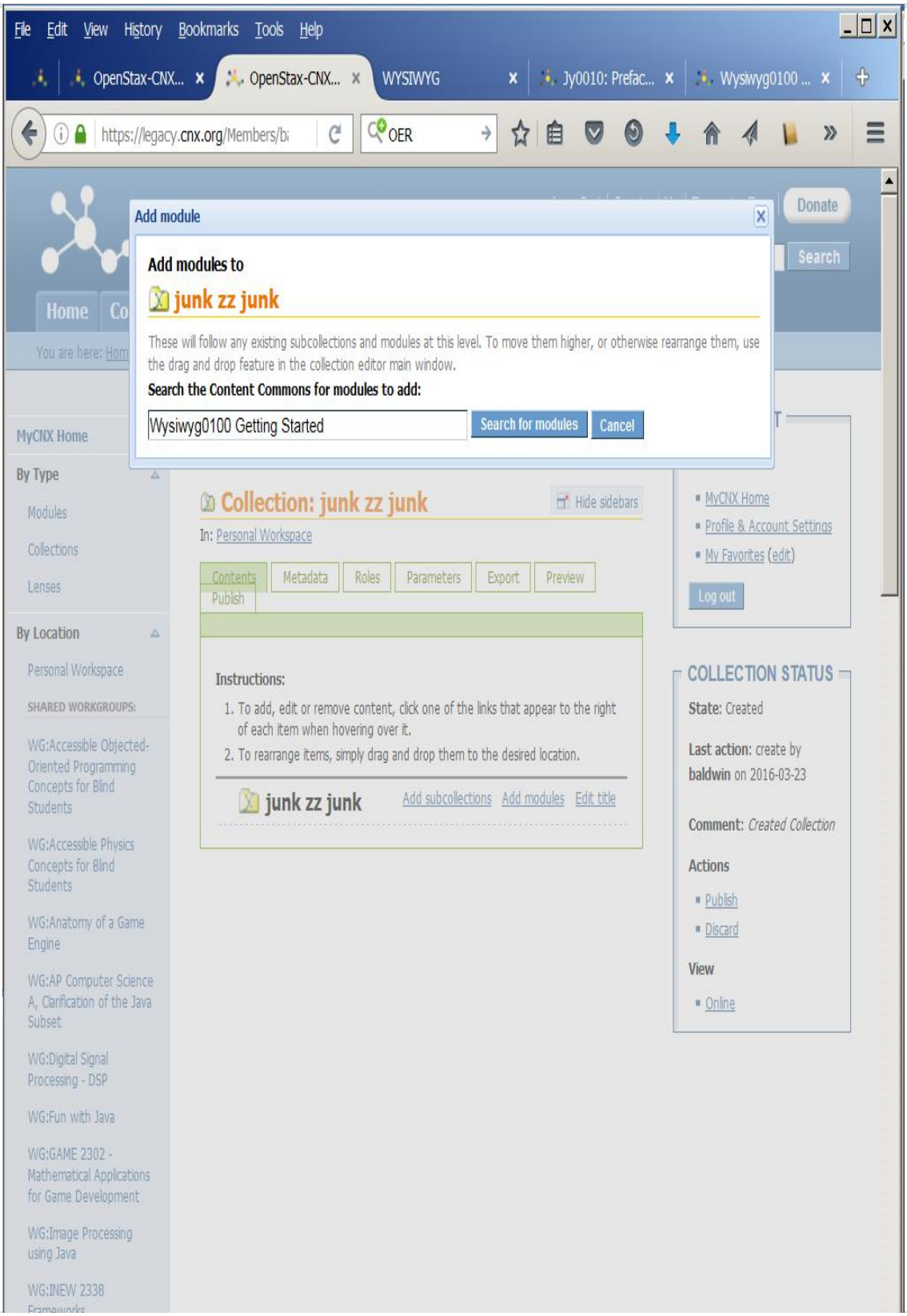
Go ahead and enter the metadata for the new **collection** and click the **Next** button.

This will cause you to land on a page that allows you to populate the **collection** with modules and perhaps subcollections as well.

Click on the link labeled **Add modules** near the center of the page *(shown in Figure 16)* and a dialog will appear at the top of the page that you can use to search for a **module** to add. Enter the title of the **module** that you want to add and you screen should look similar to that shown in Figure 16 .

Figure 16. Add modules to a collection.

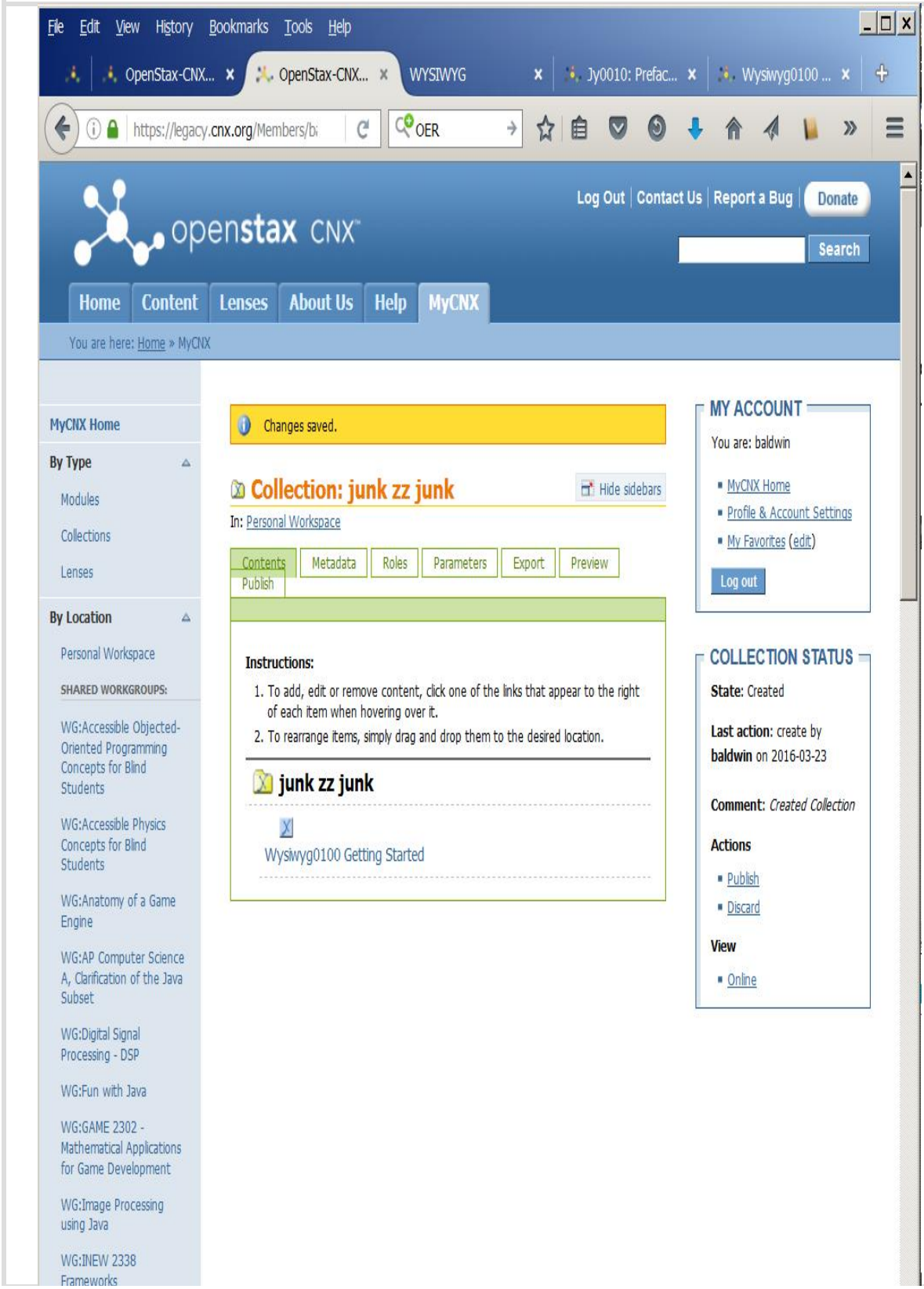
Figure 16. Add modules to a collection.



Click the button labeled **Search for modules** and after a couple of obvious steps you screen should resemble Figure 17 . Note that the new **collection** now contains the **module** that was specified for adding in Figure 16 .

Figure 17. Result of adding a module to a collection.

Figure 17. Result of adding a module to a collection.



After you have added one or more modules to your new **collection** , you will need to publish it. Select the link labeled **Publish** on the right side of Figure 17 . You will find that the steps for publishing a collections are very similar to the steps for publishing a **module** .

It is important to note that once a **module** has been added to a **collection** , if the **module** is modified, the **collection** will contain the modified version with no requirement to re-publish the **collection** . In other words, as a practical matter, the **collection** simply contains pointers to the modules that it "contains." In other words, the collections doesn't actually own the modules. Instead, the **collection** simply provides a convenient path through which the most current versions of the modules can be viewed.

Checkout a module or a collection

Once you have published a **module** or a **collection** , if you want to make changes, you must "check it out", modify it, and re-publish it. For example, Figure 12 shows a **Checkout** button that can be used to check the **module** out for modifications.

There are several ways that you can check out a **module** or a **collection** . The way that I normally do it is to open the work space that contains a list of all of the modules and collections that have been created in that work space. They are listed by title and some other information is also provided such as the current status: *created* , *modified* , *published* , etc.

Click on the title of the **module** or the **collection** that you want to modify. If the state is *created* or *modified* , it will open the page shown in Figure 7 allowing you to edit and then to re-publish the **module** or **collection** . If the state is *published* , you will get a **Checkout** button. Clicking the **Checkout** button will cause you to land on the page shown in Figure 7 .

Remember that checking out a **module** or **collection** and modifying it doesn't modify the view that is available to the public. In order to change that view, you must re-publish the modified version.

Miscellaneous

This section contains a variety of miscellaneous information.

Note: Housekeeping material

- Module name: Step-by-Step Guide to OpenStax Publishing
- File: Wysiwyg0120.htm
- Published: 03/24/16

Note: Disclaimers:
Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module. In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.
Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-

Authoring OpenStax Documents in Apache OpenOffice Writer
Learn how to create documents using Apache OpenOffice Writer and publish those documents on OpenStax.

Revised: Wed Apr 27 08:27:30 CDT 2016

This page is part of a Book titled OpenStax Publishing with a WYSIWYG Editor.

Table of contents

Preface

In an earlier module titled Wysiwyg0100 Getting Started , I showed you how you can use a WYSIWYG XHTML editor to create documents for publication on OpenStax. While that is much easier than creating your documents in OpenStax's CNXML code, it does still require the use of an editing tool that may not be familiar to many potential authors.

In this module, I will introduce you to a process and a program that will allow you to create documents using Apache OpenOffice Writer *(referred to hereafter simply as Writer)* and to publish those documents on OpenStax. With this process, you can create your documents in Writer, translate them

into CNXML using programs that I will provide, and upload them to OpenStax as Plain CNXML.

Note:
According to their website , *"Writer has everything you would expect from a modern, fully equipped word processor. It is simple enough for a quick memo, yet powerful enough to create complete books with contents, diagrams, indexes, etc. You're free to concentrate on your ideas while Writer makes them look great."*
Writer is very similar to Microsoft Word. If you are familiar with Word, you should have no difficulty making the transition to Writer.

I will create this module exclusively using Writer along with a program named **OpenOfficeToXhtml01** that I will provide. *(See Run the program .)*

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the Figures and Listings while you are reading about them.

Figures

- Figure 1 . An old caricature of the author.
- Figure 2 . A note containing an image.
- Figure 3 . The Writer hyperlink dialog box.

Listings

- Listing 1 . Stand-alone preformatted text.

- Listing 2 . A note containing preformatted text.

Preview

This is not a general purpose Writer-to-CNXML translator process. Not all Writer features can be translated into CNXML. Also, the process will not provide you with access to all of the CNXML features that are available. *(For example, it will not prepare you to create OpenStax documents having all of the features shown in the OpenStax Physics book.)* If you are looking for that degree of sophistication when publishing your documents on OpenStax, you simply need to learn how to create OpenStax documents using the online CNXML editor.

This process is designed to support a reasonable subset of the features found in Writer and the features available with CNXML. In general, you should be able to use this process to create OpenStax documents containing the features shown in this module.

Available features

Here is a list of the CNXML features that you should be able to include in your OpenStax documents using this process. These features can be created using normal Writer editing procedures. I will identify the Writer features that you can use and later translate into CNXM in this module and in several Writer sample **odt** files that I will provide. If I don't discuss a feature in this module or in one of those sample files, you probably can't use it. Those features are:

1. Headings with a parent/child relationship such as Preface , Viewing tip , and Figures shown above.
2. Ordinary paragraph text with **bold** , *Italic* , ***bold-Italic*** , etc.
3. Inserted images *(see Figure 1).*
4. Local bookmarks such as those incorporated in the headings for Preface , Viewing tip , and Figures shown above.
5. Hyperlinks to local bookmarks such as this hyperlink .
6. Hyperlinks to other websites such as the OpenStax Physics book.

7. Ordered lists such as this list.

    1. Indentation in ordered list such at this one.
    2. And this one.

8. Unordered lists such as the items under the Table of Contents .
9. Stand-alone preformatted text *(see Listing 1)* .
10. CNXML notes containing paragraph text, preformatted text, lists, and images *(see Figure 2)* .

A little more clarity

The process described in this module isn't actually a Writer-to-CNXML translator process. Instead, it is a Writer-to-XHTML translator process. Once you have created your Writer document and saved it in an **odt** file, you can use the program that I will provide to translate it into an XHTML document. That XHTML file can then be translated into CNXML and uploaded to OpenStax using the program and procedures that I explained in the earlier module titled Wysiwyg0100 Getting Started .

Note:
Note: As of April 2016, the program that I will provide with this module has only been tested under a Windows 7 operating system using Java version 8.

Special requirements

You must disable all of the **AutoCorrect** options in Writer. Those are the features that automatically convert the first word in each sentence to upper case, convert ordinary quotes into fancy quotes, convert ordinary dashes *(minus sign characters)* into fancy hyphens, etc. You can disable those

options by selecting **AutoCorrect** on the **Format** menu and then clearing the **While Typing** checkbox.

If you fail to do that, Writer will insert characters *(such as fancy quotes)* in the output that are not suitable for translating into CNXML and uploading to OpenStax.

Also, if you plan to include computer code in your document, *(see Listing 2)* , you should probably disable automatic spell checking. Otherwise, Writer may corrupt your computer code by changing the spelling of some of the code.

Discussion of available features

The CNXML features that this program supports are described below.

Headings

The program supports the use of Headings such as Preface , Viewing tip , Figures , and Listings shown above. There are probably a variety of ways to create headings in Writer. The easiest way that I have found is to:

- Highlight the text that will form the heading
- Hold down the Ctrl key
- Press the numbers from 1 through 5

The number 1 will produce the highest level heading and the number 5 will produce the lowest level heading that is apparently supported by Writer. *(You can hold the Ctrl key and press the number 6 but it won't produce a heading in the version of Writer that I am using -- 4.1.2.) For example, the headings Preface , Viewing tip , and Figures are levels 1, 2, and 3 respectively.*

Headings created in this manner will translate into h1, h2, and h3 headers in XHTML respectively. Those headers will then translate into sections in CNXML. If you are reading this document online at OpenStax, this section

titled Headings is one of many sections contained in the CNXML document.

The rules for using Headings are simple. Heading 1 is the top level heading. To produce a CNXML file that can be successfully uploaded to OpenStax, any element designated as Heading 2 must be a child of a Heading 1 element. Similarly, any element designated as Heading 3 must be a child of a Heading 2 element. You cannot create a Heading 1 element and then create a Heading 3 element below it without creating a Heading 2 element in between. *(Well, actually you can do that but it will cause problems later on when you try to upload to OpenStax.)*

See Run the program for a link to a zip file containing a sample ***odt*** *file named* ***Headings.odt*** .

Lists, paragraphs, bold, and Italic

Ordered lists

The list that you see above is an ordered list. Each item has a numeral to its left. Writer provides many options for formatting ordered lists using numerals, letters, Roman numerals, etc. However, this program supports only the simple variety that you see above using numerals that begin with the numeral "1" and increase incrementally going down the page.

Unordered lists

The list that you see under Table of contents is an unordered list, often referred to as a bullet list. There are no numbers on the left, just bullets. As before, Writer provides different options for formatting the bullets. However, this program supports only the simple variety that you see under Table of contents .

Paragraphs

This program supports ordinary Writer paragraph text. This is the type of text that you get when you start typing in a blank document. This is paragraph text that you are reading right now.

This sentence and the following sentences comprise a paragraph. In Writer, you terminate a paragraph when you press the **Enter** key. This sentence contains a **bold** word. This sentence contains an *Italic* word. This sentence contains text that is both ***bold and italic*** .

There are various ways to style text as **bold** and *Italic* . The easiest way that I know of is to

- Highlight the text that is to be styled as **bold** or *Italic*
- Hold down the Ctrl key
- Press the b-key for **bold** and the i-key for *Italic*

Note:
Important note: Because angle brackets and ampersands have very significant meanings in HTML, you cannot include angle brackets and ampersands in your Writer text unless you put them in preformatted text. If you put them in preformatted text, you must first convert them to their respective entities as shown below:

```
The entity for < is &lt;
The entity for > is &gt;
The emtity for & is &amp;
```

*See Run the program for a link to a zip file containing a sample **odt** file named **ListsParagraphsBoldAndItalic.odt** .*

Preformatted text

Preformatted text is a term that is used in HTML. To create text in Writer that will result in preformatted text in the XHTML output, you need to highlight the text and change the font to **Courier New** . *(Any text in your Writer document that is styled as Courier New will be written into the output XHTML file as preformatted text.) When preformatted text is later translated into CNXML for uploading to OpenStax, it is given a CNXML style named* ***code*** .

Here is an example of stand-alone preformatted text. A simple paragraph was used to create a caption with a bookmark that can be referenced by hyperlinks elsewhere in the document.

Listing 1. Stand-alone preformatted text.

```
//Process hyperlinks
else if(node.getNodeName() == "text:a"){
  System.out.println("---Processing text:a");
    out.println("<a href=" + "\"" +
        valueOf(node,"@xlink:href") + "\"" + ">");
  //Process all XML child nodes
  // recursively
  applyTemplates(node,null);
  out.println("</a>");
}//end else if
//End process hyperlinks
```

As you will see later in Listing 2 , preformatted text can also be put into a CNXML note, which in some cases may provide an improved visual experience in the OpenStax display.

The typical purpose of preformatted text in a web page is to allow you to preserve the format such as indentation, line spacing, etc. In this program, preformatted text is also used to include characters, such as angle brackets, that would cause problems when mixed in with HTML code.

Note:
Important notes:

1. As mentioned earlier, you may not include angle brackets and ampersands in your ordinary Writer paragraph text. However, you may put them in preformatted text provided that you first convert them to their respective entities.
2. You may not make any of the text in preformatted text into either a local bookmark or a hyperlink. You also may not apply formatting such as bold and Italic to preformatted text.

*See Run the program for a link to a zip file containing a sample **odt** file named **PreformattedText.odt** .*

Pictures, hyperlinks, and bookmarks

Pictures

You can insert images into your Writer document by selecting **Picture** on the **Insert** menu. The image shown below was inserted in that manner. Once inserted, the image becomes part of the **odt** file. It other words, the **odt** file will contain a copy of the image and not just a pointer to the original image file. Changes that you make to the image, using the **Graphic Filter** item on the **Picture Bar** , for example, will be made to the copy and not to the original image file.

Figure 1. An old caricature of the author.

An ordinary paragraph was used to create the caption with the bookmark shown above . It is best to put the caption at the top instead of at the bottom. Later when a user clicks a hyperlink to a bookmark in the caption, that bookmark will be moved to the top of the page in the browser.

For accessibility purposes, you should always provide **Alt Text** for your images. If you don't do that, OpenStax may refuse to accept and publish the image. You can create **Alt Text** by right-clicking on the image, selecting **Description** , and entering your text in the **Title** box.

Hyperlinks and bookmarks

To create a bookmark, do the following:

- Place the cursor at the location for the bookmark.
- Select **Bookmark** on the **Insert** menu
- Enter the text for the bookmark

Note:
Important: When entering the text for a bookmark, don't leave any spaces and don't use any characters other than letters, numbers, and the underscore character. A good way to create descriptive bookmarks is to use "CamelCase" to create a bookmark that looks something like the following:

```
MyVeryBestBookmark
```

To create a hyperlink to another web site, do the following:

- Highlight the text that is to become the hyperlink.
- Select **Hyperlink** on the **Insert** menu.
- Select **Internet** and enter the URL in the **Target** box.

To create a hyperlink to a bookmark in the same document:

- Highlight the text that is to become the hyperlink.
- Select **Hyperlink** on the **Insert** menu.
- Select **Document** .
- Enter the name of the bookmark in the **Target** box or click the button to the right of the **Target** box and select the bookmark in the list of **Bookmarks** .

*See Run the program for a link to a zip file containing a sample **odt** file named **PicturesHyperlinksAndBookmarks.odt** .*

Notes

A CNXML **note** is a container that can be used to cause material to be set apart from normal CNXML paragraph text providing a pleasing appearance in the OpenStax presentation.

Contents of notes

Notes can contain ordinary paragraph text, preformatted text, lists, images, or a combination of those elements. This document contains several notes that contain ordinary paragraph text such as the one shown above , which contains a mixture of ordinary paragraph text and preformatted text.

The note shown here contains a list.

The following note contains preformatted text.

Listing 2. A note containing preformatted text.

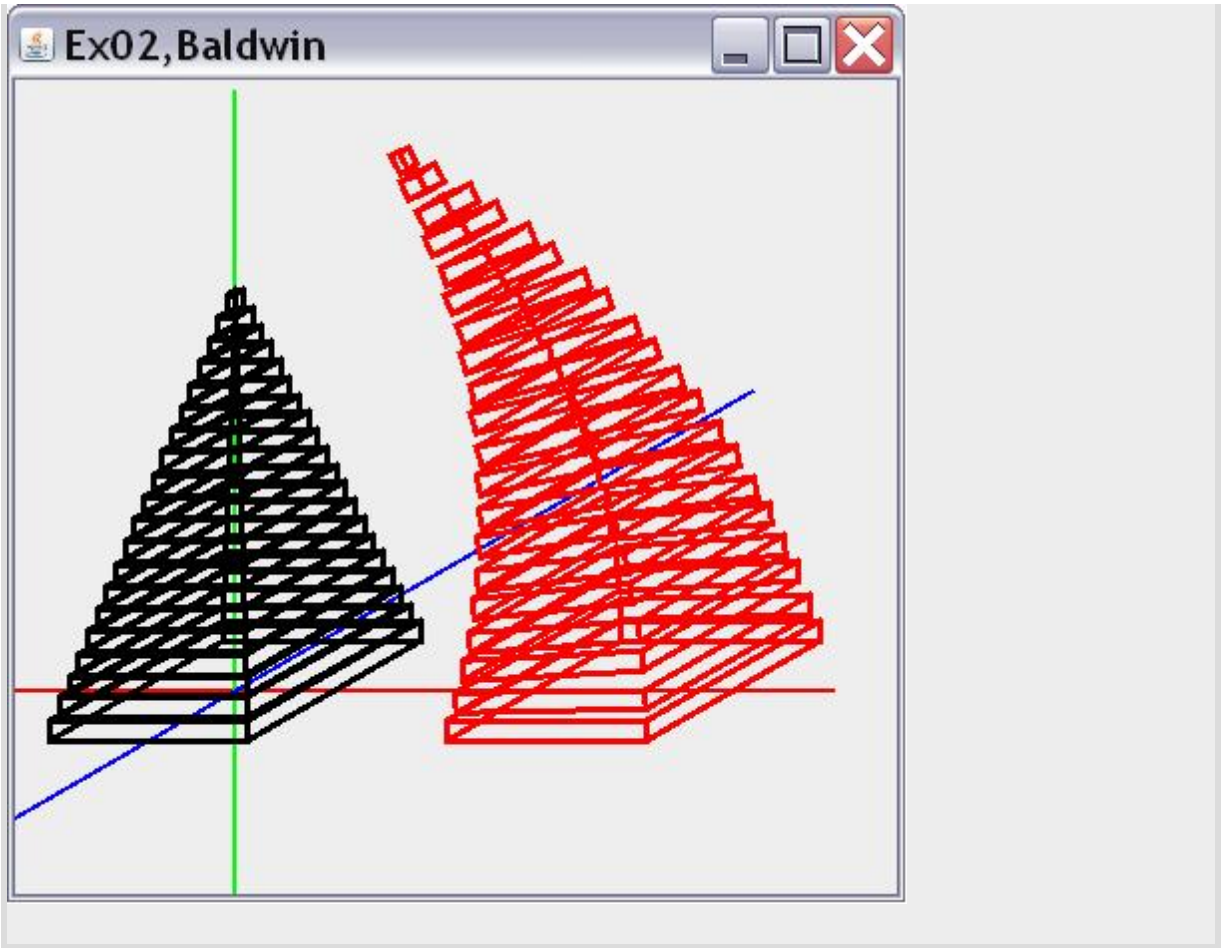
Note:

```
//Process hyperlinks
else if(node.getNodeName() == "text:a"){
  System.out.println("---Processing text:a");
    out.println("<a href=" + "\"" +
        valueOf(node,"@xlink:href") + "\"" + ">");
  //Process all XML child nodes
  // recursively
  applyTemplates(node,null);
  out.println("</a>");
}//end else if
//End process hyperlinks
```

As you can see, this is a good way to present program code. However, the visual effect might be more pleasing if the caption were moved inside the note as shown in Figure 2 .

Finally, the following note contains an image. In this case, a paragraph was used to create a caption with a bookmark **inside the note** .

Note:
Figure 2. A note containing an image.



How to create a note

To create a note in CNXML, you need to create a basic table in Writer with one row and one column. *(Don't do anything fancy when you create the table.)* You can then populate it with one or more of the items shown above. That table will eventually end up as a note on OpenStax.

Run the program

Now it's time to explain the mechanics of using this process and this program to create and publish your documents on OpenStax.

Some sample odt files

Before getting into the details, click here to download a zip file named **Samples.zip** that contains several sample **odt** files. You can use these sample **odt** files to experiment with the process and the program.

The big picture

The steps for creating and publishing your document on OpenStax are as follows:

1. Download and install the Apache OpenOffice suite. As of April 2016, it is free. Note that the suite contains several different elements in addition to **Writer** , such as **Base** , **Calc** , and **Draw** .
2. Use **Writer** to create your document as described above and save it. By default, the document will be saved in a file with an extension of **odt** .
3. Use the process described below to translate the **odt** file into an XHTML file.
4. Go to Wysiwyg0100 Getting Started/Running the program and use the procedure described there to translate your XHTML file into a CNXML file for uploading to OpenStax.
5. Go to Wysiwyg0100 Getting Started/Upload and publish your new page and use the procedure described there to upload your CNXML file and publish it on OpenStax.

The detailed picture

I will concentrate steps 3, 4, and 5 in the above list .

Although the output file that is produced when you save the document in Writer has an extension of **odt** , it is actually a zip file. You can confirm that by changing the extension from **odt** to **zip** and opening the file using whatever program you normally use to extract the contents of zip files. In Windows, for example, you can simply right-click on the zip file and select

Open in a new window . Then you can copy the contents of the zip file into a different folder.

Note:
Note that if you have a special zip program, such as WinZip installed on your computer, it will probably override the Windows default behavior described above.

Extract content.xml and Pictures

When you open the zip file for extraction, you will see that it contains a large number of files and folders. In all cases, it should contain a file named **content.xml** . If your Writer document contains pictures, the zip file will also contain a folder named **Pictures** . The **Pictures** folder will contain copies of the image files that you inserted into your document. The image files in the **Pictures** folder will probably have very long names such as

10000000000001C3000001C37E3C8FEA.jpg

This process uses only the file named **content.xml** and the folder named **Pictures** if it exists. Extract those two items into an empty folder.

Get and run the program

Click here to download a zip file named **ProgramFiles.zip** containing the compiled Java class files for the program named **OpenOfficeToXhtml01** . That zip file should contain the following four files:

- HtmlPreCleaner01.class
- OpenOfficeToXhtml01.class
- Translate01.class
- copyFilesFromPicture.bat

Extract those four files into the same folder with the file named **content.xml** . *(Note: if you haven't downloaded the program files since the revision date shown at the top of this module, it might be a good idea to download a fresh copy to make certain that you have the latest version.)*

Open a command-line window in that folder and execute the following command at the command prompt:

```
java OpenOfficeToXhtml01 content.xml
PreCleanerOutput.html > junk.txt
```

This should produce several new files in that folder including a file named **PreCleanerOutput.html** and a copy of each of the image files from the **Pictures** folder, if any.

The file named **PreCleanerOutput.html** is the XHTML file that you are looking for. If you open that file in your browser, you will see an XHTML version of your document.

Translate XHTML into CNXML

Copy that file along with the image files into another empty folder. Then go to Wysiwyg0100 Getting Started/Running the program and follow the procedures given there to translate the XHTML file named **PreCleanerOutput.html** into a CNXML file.

As described in that module, you will open a command-line window in that folder and execute the following command at the command line:

```
java CNXMLprep12 PreCleanerOutput.html
OutputFile.cnxml
```

where

- **PreCleanerOutput.html** is the name of the input file.
- **OutputFile.cnxml** is the name that you want to assign to the CNXML file produced by the program. This is the file that you will upload to

Upload and publish the CNXML file

Then go to Wysiwyg0100 Getting Started/Upload and publish your new page and follow the instructions there to upload your CNXML file to OpenStax, and to publish it on OpenStax.

Note:
This program was developed using the Java SE Development Kit 8 . In order to run this program on your computer, you must have the Java SE Runtime Environment 8 (JRE) or a later version of the JRE installed on your computer. The JRE is already installed on many computers and it may already be installed on yours *(but it may be an earlier version that requires updating)* . If it is not installed, or it is not up to date, there are many web pages that provide instructions for downloading and installing Java. One of those web pages is located here .

Downloadable files

There is at least one major area where Writer doesn't serve us well for creating documents that will be published on OpenStax. Therefore, it was necessary for me to develop a workaround to deal with the issue.

The issue is that as in the case in Get and run the program above, we often want to make it possible for the user to download a zip file, an audio file, or some other file type *(other than an image file)* by clicking on a link in the OpenStax document. As near as I can tell, Writer doesn't provide direct support for that capability.

A somewhat unique code

Before getting into the details, let me introduce you to a somewhat unique code that is an integral part of my workaround:

```
12345OIcafa678910TIliga
```

I say it is somewhat unique because I didn't find any matches for the code with a Google search.

To make the code easy to remember, the characters in the code are the numeric and letter equivalents of the first letters of the words in the following nursery rhyme:

```
One, two, three, four, five,
Once I caught a fish alive,
six, seven, eight, nine, ten,
Then I let it go again.
```

How to use the code

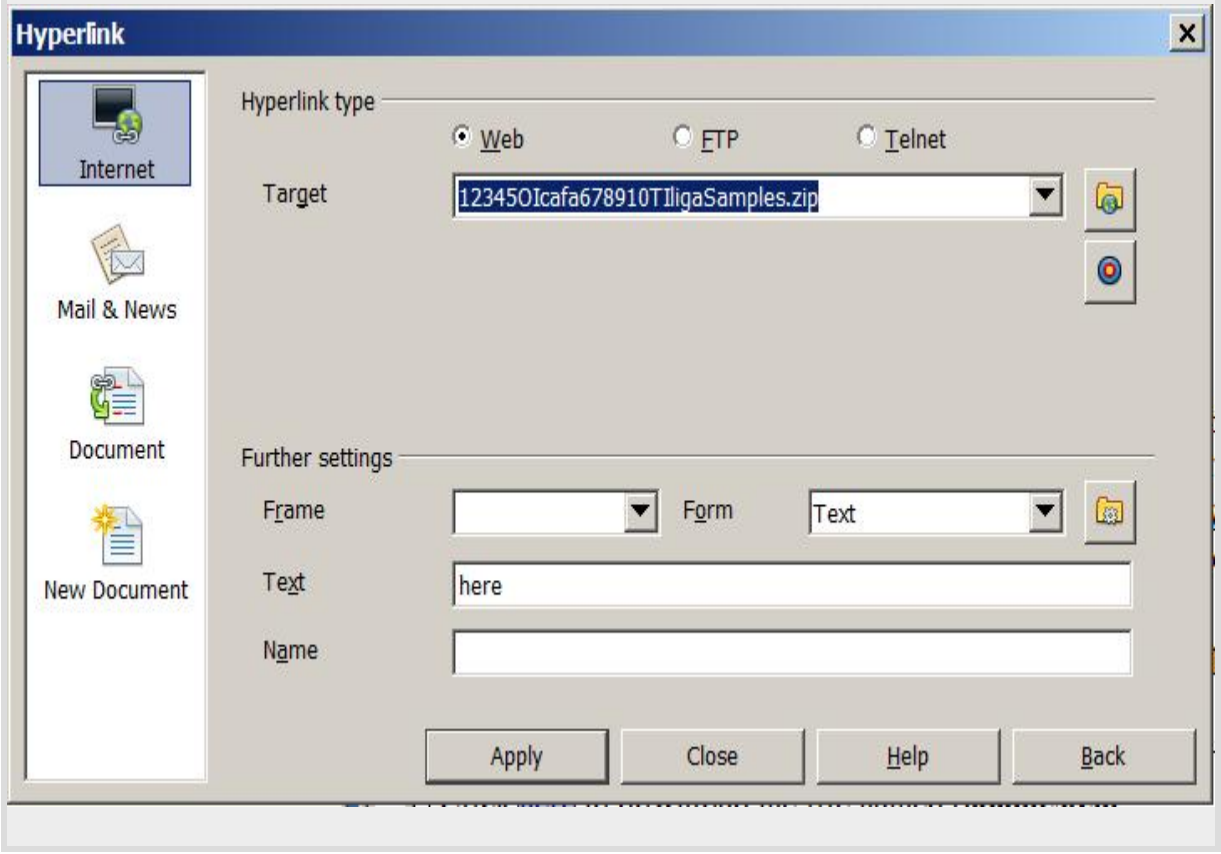
Now that you know what the code is and how to remember it, I will explain how to use it. Assume that you want to make it possible for your document's viewers to download a file named **Samples.zip** from the OpenStax site. *(You will have placed that zip file along with any image files and other downloadable files in a zip file and uploaded it to OpenStax as described here and here .)*

Highlight the text in your document that will be the hyperlink for downloading the file, such as the word **here** in the following sentence:

Note:
Click here to download the file named **Samples.zip** .

Pull down the **Insert** menu and select **Hyperlink** . The dialog box shown in Figure 3 will appear.

Note:
Figure 3. The Writer hyperlink dialog box.



Select **Internet** on the left, check **Web** at the top and type the code into the **Target** field followed immediately by the name of the downloadable file. Then click the **Apply** button and the **Close** button.

This will cause a hyperlink pointing to a website with the name shown in Figure 3 to be contained in the file that is first produced by translating the file named **content.xml** into XHTML. That is not what we are looking for. What we are looking for is a hyperlink to a file named **Samples.zip** that is in the same folder as the CNXML file on the OpenStax server. The

hyperlink is modified to the following form for the final XHTML output file. This is exactly what we need.

```
<a href="Samples.zip">
```

Clicking that link will cause the file named **Samples.zip** to be downloaded.

Miscellaneous

This section contains a variety of miscellaneous information.

Note:
Housekeeping material

- Module name: Authoring OpenStax Documents in Apache OpenOffice Writer
- File: Wysiwyg0140.odt
- Published: 04/23/16

Note:
Disclaimers:
Although the OpenStax site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.
I also want you to know that, I receive no financial compensation from the OpenStax website even if you purchase the PDF version of the module.
In the past, unknown individuals have copied my modules from OpenStax, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available

on OpenStax and that it was made and published without my prior knowledge.

-end-